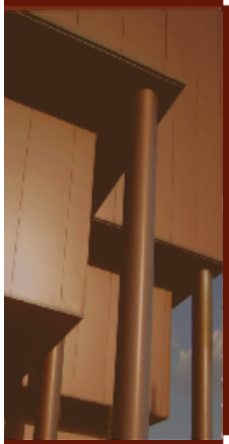
Trabajo Fin de Grado

Grado en Ingeniería en Tecnologías Industriales

# Plataforma de telecontrol de un drone comercial

Autor: Emilio Marín Fernández

Tutor: José María Maestre Torreblanca



**Departamento de Ingeniería de Sistemas y Automática**
**Escuela Técnica Superior de Ingeniería**

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

# Plataforma de telecontrol de un drone comercial

Autor:
Emilio Marín Fernández

Tutor:
José María Maestre Torreblanca
Profesor Contratado Doctor

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Grado: Plataforma de telecontrol de un drone comercial

Autor: Emilio Marín Fernández

Tutor: José María Maestre Torreblanca

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

# Agradecimientos

*A mis padres, hermanos y novia por su apoyo incondicional.*

*A mi abuelo por todo lo que me ha enseñado.*

*A todos los profesores por su entusiasmo a la hora de dar clase.*

*A mi tutor Jose María Maestre por toda la ayuda prestada durante la realización de este proyecto.*

*Emilio Marín Fernández*

*Grado en Ingeniería de Tecnologías Industriales,*

*Sevilla, 2017*

# Resumen

En el presente proyecto se realiza el análisis completo de una librería desarrollada por el *MIT* que permite el diseño y simulación de sistemas de control para el minidrone Parrot Rolling Spider a partir de *Matlab* y *Simulink.*

Se describen los pasos a seguir para la puesta en marcha del quadrotor, desde el software necesario hasta los comandos necesarios para el primer vuelo del drone.

Para el control del quadrotor, se diseñan tres sistemas de control basados en los conocimientos adquiridos durante el paso por la universidad, se han escogido los controladores: PID, LQR y el controlador por asignación de polos. Estos se someten a múltiples experimentos para encontrar el que da una respuesta mejor.

Una vez escogido el controlador óptimo, se desarrollan aplicaciones para ampliar las funcionalidades iniciales del drone, consiguiendo el reconocimiento de una plataforma de aterrizaje analizando las imágenes tomadas por el vehículo y el envío de comandos al drone con la voz, utilizando una librería externa.

Por último, se presentan futuros proyectos que podrían realizarse sentando sus bases en este, desarrollando aplicaciones más complejas y mejorando el control del drone.

# Abstract

In the current project, a complete analysis of a toolbox developed by the MIT is realized. This allows the design and the simulation of control systems for the Parrot Rolling Spider minidrone based on *Matlab* and *Simulink*.

Steps are given for the setting up of the quadrotor, from the necessary software to the commands required for the first drone flight.

For the control of the quadrotor, there are three control systems designed, based on the knowledge acquired in the university. The controllers that have been chosen are: PID, LQR and Pole Placement controller. These are subjected to multiple experiments to find those that give better response.

Once the controllers are designed, applications are developed to increase the initial functionalities of the drone e.g., to recognize a landing platform analyzing the images taken by the vehicle and using voice commands to control the drone by means of an external *API*.

Finally, future projects could be carried out being based on this one, developing more complex applications and improving the control of the drone.

# Índice

# ÍNDICE DE TABLAS

# ÍNDICE DE FIGURAS

# Notación

| | |
|---|---|
| $\phi$ | Giro alrededor del eje X, Pitch/Cabeceo. |
| $\theta$ | Giro alrededor del eje Y, Roll/Alabeo. |
| $\psi$ | Giro alrededor del eje Z, Yaw/Guiñada. |
| | **Filtro de Kalman** |
| $\hat{x}^-_k$ | Predicción del vector de estado en el instante k. |
| $A$ | Matriz del sistema lineal. |
| $B$ | Matriz del sistema lineal. |
| $u_k$ | Entrada al sistema en el instante k. |
| $\hat{x}_k$ | Estimación del vector de estado en el instante k. |
| $P^-_k$ | Predicción de la covarianza del error a priori. |
| $P_k$ | Estimación de la covarianza del error. |
| $Q$ | Varianza del error del proceso. |
| $R$ | Varianza del error de medida. |
| $K_k$ | Matriz de ganancia de Kalman. |
| $z_k$ | Valor medido en el instante k. |
| $I$ | Matriz identidad. |
| | **Controlador PID** |
| $u$ | Señal de control. |
| K | Ganancia proporcional. |
| $T_i$ | Tiempo integral. |
| $T_d$ | Tiempo derivativo. |
| $e_k/e(t)$ | Error cometido en el instante k / dependiente del tiempo. |
| | **Controlador LQR** |
| A,B,C | Matrices del sistema lineal. |
| $u$ | Entrada del sistema. |
| $J_{LQR}$ | Función coste a minimizar. |
| Q | Matriz de ponderación de estados. |
| R | Matriz de ponderación de control. |
| $\rho$ | Parámetro de ajuste entre control y estados. |
| L | Función Lagrangiana. |
| $\lambda$ | Multiplicador de Lagrange. |
| H | Función Hamiltoniana. |

| | |
|---|---|
| $\alpha_i$ | Peso de la componente i de estados. |
| $\beta_i$ | Peso de la componente i de control. |
| $(x_i)_{max}$ | Valor máximo del estado i. |
| $(u_i)_{max}$ | Valor máximo de la componente i de control. |

**Controlador por asignación de polos**

| | |
|---|---|
| FSF | Full-State Feedback controller. |
| A,B,C | Matrices del sistema. |
| $\lambda$ | Autovalores. |
| I | Matriz identidad. |
| $A_{BC}$ | Matriz del sistema en bucle cerrado. |
| K | Ganancia. |
| J | Forma canónia de Jordan. |
| V | Matriz de cambio de base. |

# 1 INTRODUCCIÓN

*La ciencia puede divertirnos y fascinarnos,*

*pero es la ingeniería la que cambia el mundo.*

Isaac Asimov, 1920-1992

En este documento, se realiza un análisis completo de la librería diseñada en el Massachusetts Institute of Technology (*MIT)* que permite diseñar y simular algoritmos de control y estimación para el drone Parrot Rolling Spider, con la posibilidad de generar el propio código en C integrado para subirlo al vehículo y realizar vuelos reales de los experimentos elegidos. Para la implementación se utiliza *Matlab* y *Simulink*, *Matlab* es una herramienta de software matemático que cuenta con un lenguaje propio, aunque para este proyecto se realizará la mayoría de la programación en C. *Simulink*, por otro lado, es un entorno de diagramas de bloques para la simulación y el diseño basado en modelos, este se integra con Matlab y permite definir el comportamiento de los distintos modelos creados a partir de funciones [1]. Todos los archivos de código utilizados a lo largo de este proyecto y vídeos pueden ser encontrados en el repositorio de Github del proyecto [2].

La estructura de este proyecto se puede dividir en cuatro bloques.

En primer lugar, se da una descripción del drone que se utiliza, con especificaciones, software necesario para poder empezar a usar la librería y las ecuaciones básicas del modelo matemático que intervienen en la dinámica de un drone.

En el segundo bloque se realiza la puesta en marcha de la librería, paso a paso se instalan todos los componentes necesarios para conseguir una primera simulación de vuelo del quadrotor. En este apartado también se mencionan todos los archivos de código importantes, aquellos aptos a ser modificados para conseguir un mejor funcionamiento o una nueva habilidad del drone.

En el siguiente bloque, se proponen tres estructuras de control, a partir de los conocimientos adquiridos durante el paso por la universidad, que se pondrán a prueba con diferentes experimentos, obteniendo gráficas que ayudarán a decidir que sistema de control es el proporciona mejores resultados.

En el último bloque, se desarrollan algunas aplicaciones interesantes que llevan más allá la capacidad del drone, haciendo uso de la cámara que incorpora para la detección de objetos o utilizando librerías externas para añadir nuevas funcionalidades.

Seguidamente, finalizado el desarrollo de estos bloques principales, se dan una serie de conclusiones alcanzadas durante el progreso de este proyecto y se piensan posibles aplicaciones futuras que se podrían añadir a este drone, dando paso a futuros trabajos que puedan sentar sus bases en este documento.

## 1.1 Motivaciones

En los últimos años se han observado grandes avances en las tecnologías de los *Unmanned Aerial Vehicle* (UAV) que han popularizado su uso fuera del sector militar. Se está apostando fuertemente por innovación para la utilización de vehículos aéreos dentro del área de servicios o la industrial para poder automatizar algunas actividades que por diferentes razones pueden ser costosas o peligrosas para el ser humano. A través del telecontrol de quadrotors se puede acceder con facilidad a lugares a los que una persona gastaría numerosos recursos y tiempo en llegar o que incluso sería imposible el acceso debido a las condiciones que presente, altos niveles de radiación o contaminación, con un operario controlando el vehículo desde su cuadro de mandos podría, por ejemplo, inspeccionar anomalías en infraestructuras de difícil acceso en un tiempo record.

Otro paso más dentro del control de estos vehículos aéreos aparece con la llegada de los sistemas autónomos, las mismas tareas en las que sería necesario a un operador controlando al vehículo pasan a ser completamente independientes, de esta forma, por ejemplo, se puede estar sobrevolando una extensión de terreno según una trayectoría preprogramada para la detección de incendios y poder así actuar lo más rápido posible.

Estas y muchas otras aplicaciones dentro del sector industrial son las que me motivan para la realización del proyecto elegido para poder así crear distintas aplicaciones de control autónomo que podrían ser aplicadas a sistemas de mayor potencia al utilizado.

## 1.2 Definiciones y tipos de UAV

El término UAV viene del ingés *Unmanned Aerial Vehicle* que significa Vehículo Aéreo No Tripulado, este incluye tanto a los vehículos que son controlados remotamente por un piloto, los *Remotely Piloted Aircraft* (RPA), como a los que contienen un sistema autónomo. En este caso, tomamos como referencia el segundo concepto, ya que el objetivo es el de generar un control independiente de la acción de una persona por lo que se podrá también utilizar el término *Unmanned Aircraft System* (UAS).

Se puede realizar una clasificación de los UAV según su tipo de despegue y de ala.

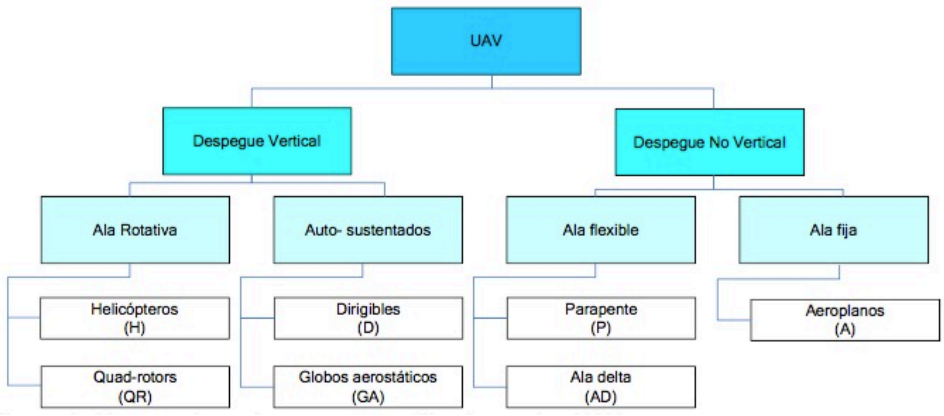



**Figura 1-1. Clasificación según tipo de aeronave** [3]

Según esta primera clasificación, el vehículo comercial utilizado para este proyecto posee cuatro alas rotativas y un despegue vertical, por lo que entraría dentro de los Quad-rotors.

Otra clasificación a tener en cuenta relaciona la altitud que el vehículo puede tomar y su duración de vuelo máxima.

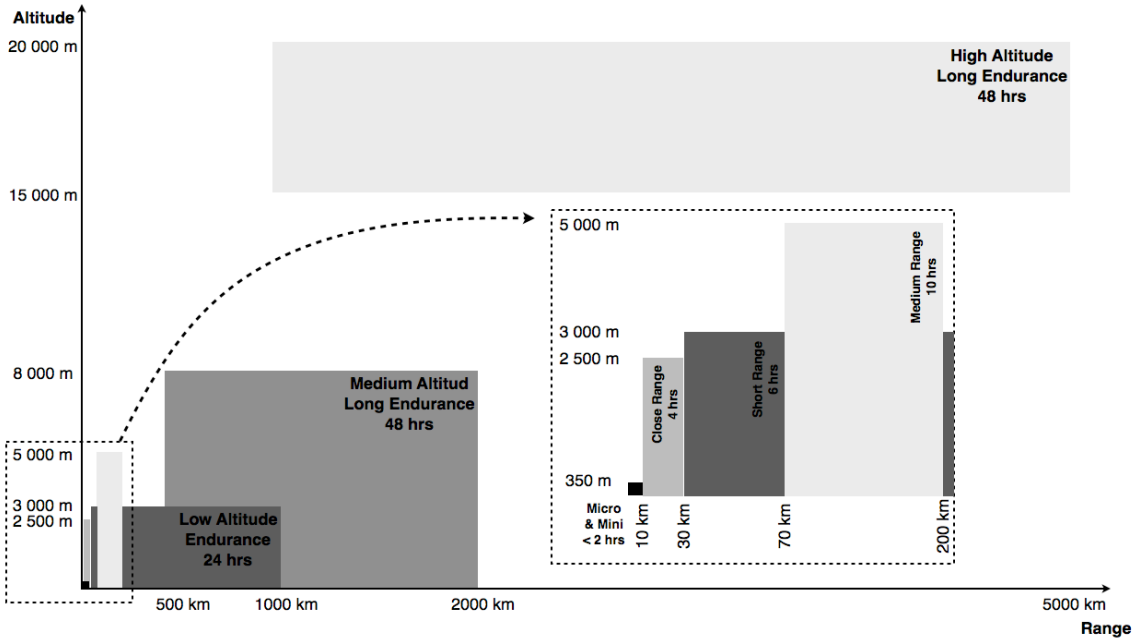



**Figura 1-2 Clasificación según la OTAN** [3]

El vehículo utilizado se encontraría dentro de la categoría más cercana al punto de origen, la micro y mini, ya que posee una duración de batería de unos 8-15 minutos y llega a una altura de unos 20-40 metros.

## 1.3 Historia de los UAV

La documentación sobre la historia de los UAV se ha extraído de [4], [5] y [6]. A continuación se exponen una serie de hechos históricos que han marcado la evolución de lo sistemas aéreos.

- La primera constancia de lo que se puede denominar UAV aparece en 1863 con la patente US 37771 A [7], la cual establece un sistema de bombardeo con globos aerostáticos cargados usados para atacar la ciudad de Venecia. En la Figura 1-3 se muestra el mecanismo usado para soltar el explosivo. Globos bombarderos con mecanismos a este se utilizaron ampliamente durante la Guerra Civil Estadounidense (1861-1865).

- Durante la Primera Guerra Mundial (1914-1918) se desarrollaron numerosos aviones no tripulados como el Kattering Bug (Figura 1-3) que contaría revoluciones del motor hasta llegar a la previamente establecida y entraría en parada actuando como un torpedo hacía el blanco. Este nunca se usó en combate.

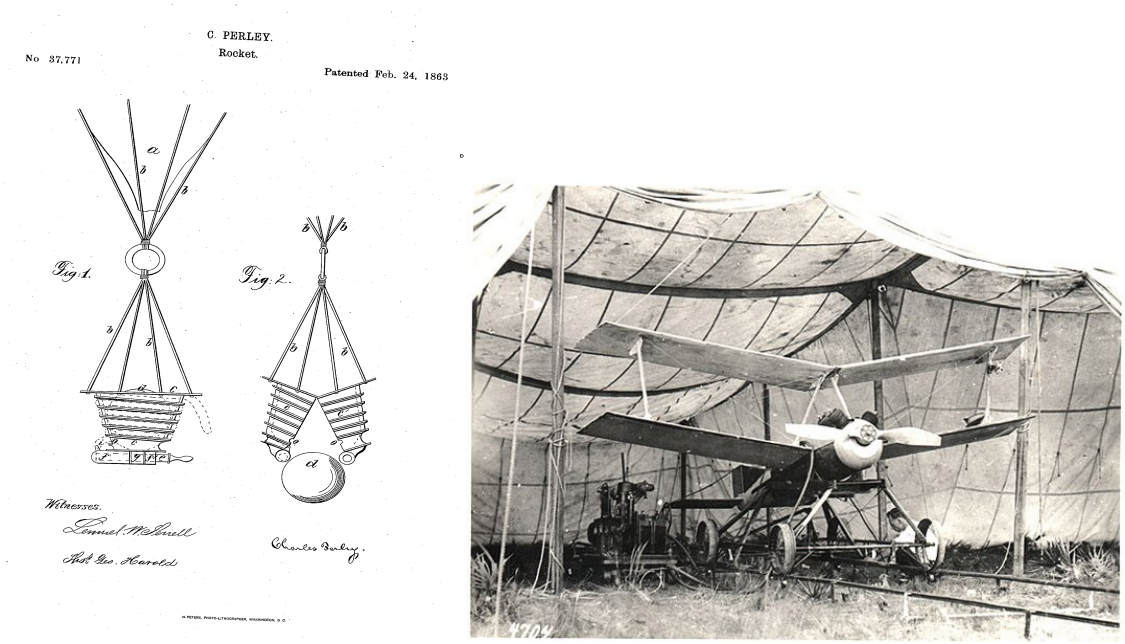



**Figura 1-3 Sistema de Bombardeo Perley** [7]**, 1863 y Kattering Bug** [4]**, 1918**

- En la Segunda Guerra Mundial, tanto los alemanes como los aliados, investigaron el campo de los UAV para fabricar misiles teledirigidos. Un ejemplo de estos UAV es el que se muestra en la Figura 1-4, fabricado por los alemanes para bombardear zonas no militares de Gran Bretaña. Fue llamado V-1 «Buzz Bomb» por el peculiar sonido que realizaba.

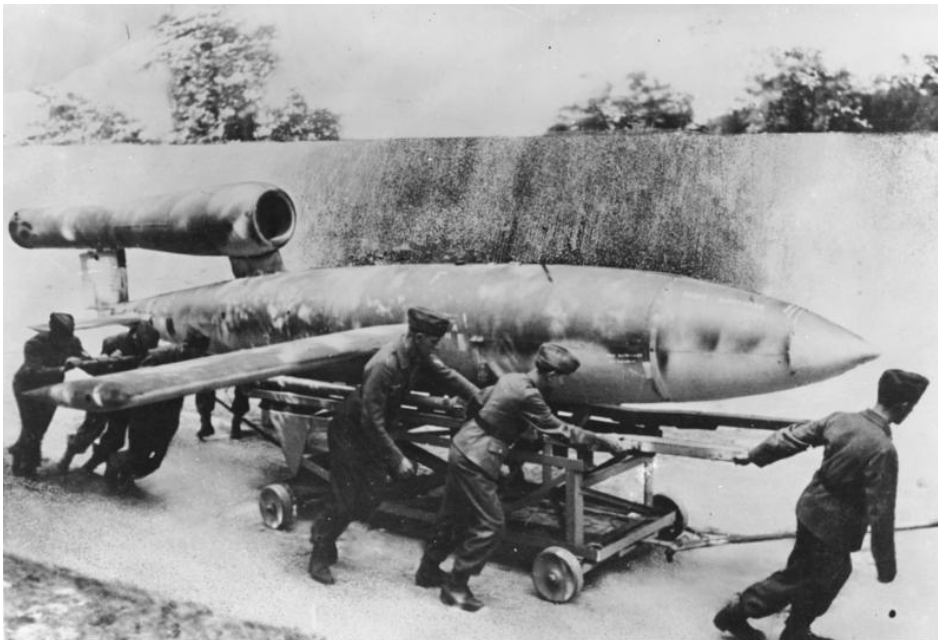


**Figura 1-4 Bombardero V-1 "Buzz Bomb"** [4]

- Pasada la Segunda Guerra Mundial, se dejo de lado el desarrollo de UAV con el fin de la destrucción y se pasó al espionaje. Se crearon diferentes sistemas de reconocimiento montados en aviones sin piloto que podían volar durante muchas horas y a grandes alturas para no ser descubiertos, así podían tomar fotografías de las zonas que sobrevolaran y transmitirlas a la base de control para su análisis.

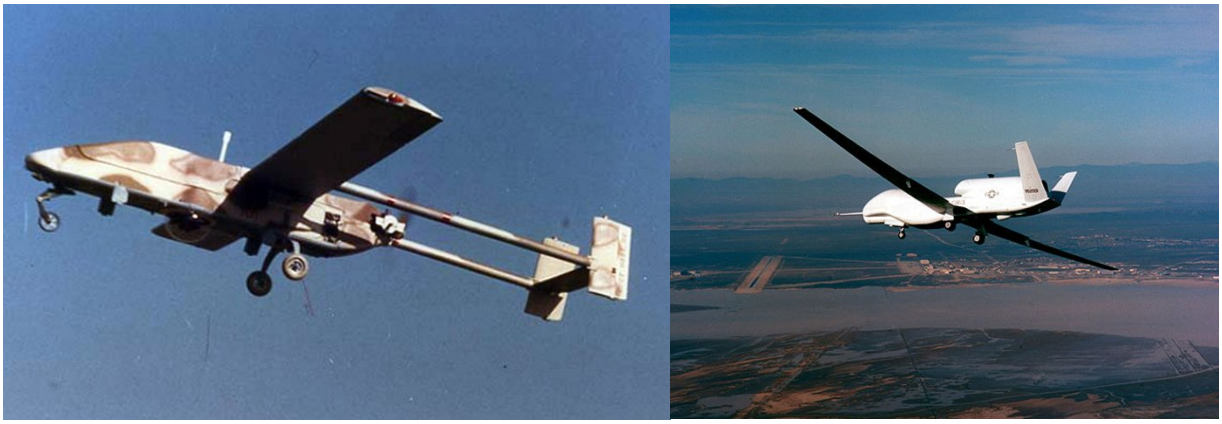


**Figura 1-5 UAV de Reconocimiento, 1970 y 1995** [4]

- En los últimos años, estas tecnologías se han acercado más al uso civil, pero siguen sin estar totalmente implantadas. Ha aparecido una variación de los UAV hasta ahora mencionados que siguen una estructura más similar a la de un helicóptero y que pueden tener un mayor número de hélices, lo cual mejora su potencia y la cantidad de peso que puede levantar. Algunas de las aplicaciones para las que se utilizan UAV como el de la Figura 1-6 son las siguientes:
    - Inspección de infraestructuras y grandes obras civiles.
    - Detección de incendios forestales.
    - Fotografía y toma de vídeos aéreos.

**Figura 1-6 Drone comercial con cámara** [4]

# 2 DESCRIPCCIÓN DEL ROBOT

En este capítulo se realizará una descripción exhaustiva del drone seleccionado para el proyecto. Toda la información mostrada se puede encontrar en diferentes fuentes como: [8] y [9].

El drone escogido se fabrica en la empresa Parrot, fundada en 1994 y actualmente es uno de los primeros fabricantes de manos libres del mundo. En 2010 introdujeron a su línea de productos su primer drone, el AR.Drone. Desde entonces, han fabricado varias versiones de ese primer vehículo aéreo, así como otros modelos, en concreto una serie de minidrones de precio y tamaño reducido. En particular, el drone elegido es el Parrot Rolling Spider, debido a su bajo coste, posibilidad de volar en interiores y a la buena documentación que existía a la hora del comienzo del proyecto, esto último se tratará en el capítulo 3 del proyecto.

## 2.1 Especificaciones

A continuación, se hablará de los distintos componentes y especificaciones que posee el drone.

- Posee unas dimensiones de 140x140x38.1 mm y un peso de 65 gramos.
- Utiliza baterías de tipo LiPo de 550mAh y 3.7V. Según las especificaciones que da Parrot, la batería tarda en cargarse 25 minutos, pero en la práctica se acerca más a una hora.
- Velocidad máxima de 4.9m/s, por lo que es una velocidad bastante baja comparada con otras opciones del mercado, pero para el alcance que se quiere obtener es perfecto.

**Figura 2-1 Parrot Rolling Spider** [8]

- Tiene una cámara vertical situada en la parte inferior del drone con una resolución de 0.3 megapíxeles, 640x480. Puede grabar hasta 60 frames por segundo. En la Figura 2-2 se puede observar esta cámara.
- La conexión al drone se puede realizar desde una aplicación proporcionada por Parrot, [10] y [11], o bien desde un ordenador con Linux como se utilizará más adelante. Todo ello se realizará a través de Bluetooth 4.0 o Low Energy (*BLE*). Esta conexión BLE nos permite volar el drone hasta un máximo de 20 metros.
- En la parte inferior de cuerpo tiene un sensor ultrasónico que se usa para medir las distancias hasta el suelo. También contiene en el interior un acelerómetro y giroscopio, en concreto el MPU6050, y un barómetro, combinando todos estos elementos, el drone es capaz de realizar vuelos estables. En la

Figura 2-2 se puede observar también el sensor ultrasónico.

- La placa base se encarga del control de estabilización y piloto automático del Parrot Rolling Spider. Incluye el chipset Parrot SIP6 con un procesador ARM A9 de 800MHz. Funciona con Linux y tiene 1Gb de memoria RAM a 256MB DDR
- Posee un par de LEDs delanteros que actúan como indicadores para mostrar alguna información o alerta que el drone quiera proporcionar.

**Tabla 1 Comportamiento de LEDs**

| Comportamiento LEDs | Significado |
|---|---|
| LED derecho en rojo. | Drone cargando, se apaga cuando está completamente cargado. |
| Parpadeo rápido verde y rojo. | Drone va a despegar. |
| Ambos LEDs en rojo. | Batería baja. |
| LEDs en verde. | Conexión por Bluetooth realizada con éxito. |

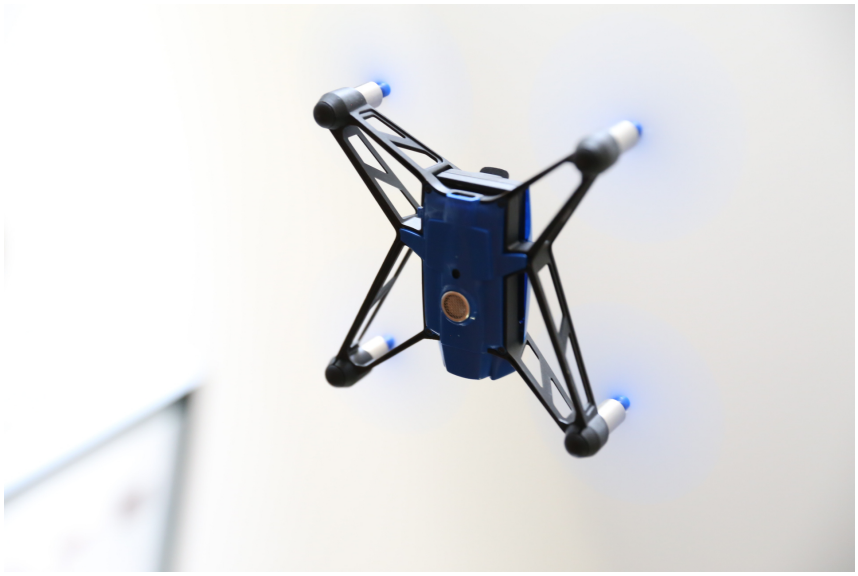


**Figura 2-2 Vista inferior del Drone** [8]

- Otros elementos muy útiles son unas ruedas de protección que consiguen evitar que el drone sufra directamente un choque, cosa que ocurrirá durante las pruebas de los distintos controladores y aplicaciones que se desarrollen. Este sistema se puede observar en la Figura 2-3.

**Figura 2-3 Drone con ruedas de protección** [8]

- El drone tiene 4 motores eléctricos, dos giran en sentido horario y los otros dos en sentido anti horario. Estos motores tienen unas dimensiones de 20x8.5 mm y un peso de 4.9 gramos. Voltaje de funcionamiento: 1.5-3.7V.

## 2.2 Software necesario

Para la conexión con el Parrot Rolling Spider se hace uso de una máquina virtual con Ubuntu 14.04.4 de 64 bits. La decisión de usar este sistema operativo viene marcada por el toolbox que se utiliza, ya que este está probado y tiene el funcionamiento asegurado en este sistema. Los detalles de las herramientas que se utilizan se describirán en el capítulo 3. La imagen ISO de Ubuntu se ha descargado desde su página oficial [12] y se ha instalado en la máquina virtual VMWare Fusion para Mac OSX [13].

Una vez instalado y configurado, es necesario instalar una versión de MATLAB que contenga Simulink. En este caso se utiliza la versión R2015a.

Para poder realizar la conexión por bluetooth, es recomendable utilizar un adaptador de BLE, en el caso de que el ordenador no tenga un chip con esas características. Aquí se utiliza el adaptador Trust 18187, de coste reducido y asegurado para funcionar bajo Linux.

## 2.3 Modelado

### 2.3.1 Introducción

Los quadrotor, como ya se ha comentado, cuentan con dos pares de motores que giran en sentido opuesto. Esto ayuda a compensar el par total del sistema. En la Figura 2-4 se muestra un esquema de los giros que intervienen en el modelo y los momentos que producen cada motor.

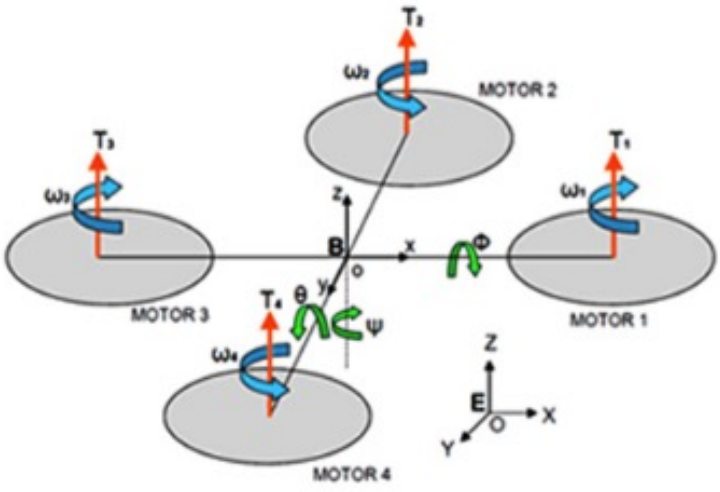



**Figura 2-4 Esquema de giros y momentos** [14]

Según si se quiere que el drone modifique algún ángulo de giro o que se desplace en alguna dirección, habrá que mandar ciertas señales a los motores, por ejemplo:

- Para conseguir un desplazamiento en el eje X, el drone tendrá que encontrarse con un ángulo de cabeceo o pitch, $\boldsymbol{\theta}$, determinado y entonces mantener la velocidad de los cuatro motores constante para no generar momento en dirección vertical y así habrá únicamente un empuje según la componente horizontal.
- Por otro lado, para conseguir ese ángulo de cabeceo, será necesario mantener constante e igual la velocidad de los motores 2 y 4 y hacer rotar los motores 1 y 3 a velocidades diferentes, lo que provocará un desequilibrio de fuerzas en el eje X.

### 2.3.2 Modelo matemático del quadrotor

En primer lugar, se define la notación que se usará a lo largo del proyecto [15]:

- **P** representa las coordenadas XYZ inerciales desde el centro de gravedad del drone

$$\boldsymbol{P} = \begin{pmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \\ \boldsymbol{Z} \end{pmatrix}$$

- $\dot{\boldsymbol{p}}$ es un vector que contiene las velocidades lineales del quadrotor según el sistema de coordenadas mundo.

$$\dot{\boldsymbol{p}} = \begin{pmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{y}} \\ \dot{\boldsymbol{z}} \end{pmatrix}$$

- **O** representa los ángulos de Euler (yaw, pitch, roll).

$$\boldsymbol{O} = \begin{pmatrix} \boldsymbol{\psi} \\ \boldsymbol{\theta} \\ \boldsymbol{\phi} \end{pmatrix}$$

- $\boldsymbol{W^{-1}}$ relaciona las rotaciones angulares del cuerpo en los ejes xyz, $\dot{\boldsymbol{o}}$.

$$\dot{\boldsymbol{o}} = \begin{pmatrix} \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{r}} \end{pmatrix}$$

a rotaciones según los ángulos de Euler, $\dot{\boldsymbol{O}}$.

$$\dot{\boldsymbol{O}} = \begin{pmatrix} \dot{\boldsymbol{\psi}} \\ \dot{\boldsymbol{\theta}} \\ \dot{\boldsymbol{\phi}} \end{pmatrix}$$

- **T** es el empuje total que generan los motores en cada eje expresado en el cuerpo del drone. $\tau_{yaw}$ , $\tau_{pitch}$ y $\tau_{roll}$ son los pares de torsión producidos por la resistencia de las hélices y la aceleración angular en los ejes z,y, x respectivamente.
- **G** es el vector de fuerza de la gravedad.
- **J** representa la inercia del drone.
- El vector de variables de estado, **x**, será:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{P} \\ \boldsymbol{O} \\ \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{o}} \end{bmatrix}$$

- **m** representa la masa del quadrotor.
- Se denomina matriz de rotación a aquella que representa una rotación en el espacio euclídeo, de esta forma se definen las rotaciones según los ejes xyz a partir del ángulo de Euler que se desea girar.

$$\boldsymbol{R_x} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{cos\phi} & \boldsymbol{sin\phi} \\ \mathbf{0} & \boldsymbol{-sin\phi} & \boldsymbol{cos\phi} \end{bmatrix}$$

$$\boldsymbol{R_y} = \begin{bmatrix} \boldsymbol{cos\theta} & \mathbf{0} & \boldsymbol{-sin\theta} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \boldsymbol{sin\theta} & \mathbf{0} & \boldsymbol{cos\theta} \end{bmatrix}$$

$$\boldsymbol{R_z} = \begin{bmatrix} \boldsymbol{cos\psi} & \boldsymbol{sin\psi} & \mathbf{0} \\ \boldsymbol{-sin\psi} & \boldsymbol{cos\psi} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Se puede realizar una secuencia de rotaciones aplicando una matriz de rotación resultante:

$$\boldsymbol{R_F = R_x * R_y * R_z}$$

$$\boldsymbol{R_F} = \begin{bmatrix} \boldsymbol{c\theta c\psi} & \boldsymbol{c\theta s\psi} & \boldsymbol{-s\theta} \\ \boldsymbol{s\phi s\theta c\psi - c\phi s\psi} & \boldsymbol{s\phi s\theta s\psi + c\phi c\psi} & \boldsymbol{s\phi c\theta} \\ \boldsymbol{c\phi s\theta c\psi + s\phi s\psi} & \boldsymbol{c\phi s\theta s\psi - s\phi c\psi} & \boldsymbol{c\theta} \end{bmatrix}$$

De esta forma se obtiene la dinámica final del quadrotor:

$$\dot{\boldsymbol{P}} = \boldsymbol{R_F}^{\boldsymbol{T}} \cdot \dot{\boldsymbol{p}}$$

$$\dot{\boldsymbol{O}} = \boldsymbol{W^{-1}} \cdot \dot{\boldsymbol{o}}$$

$$\ddot{\boldsymbol{p}} = \boldsymbol{R_F} \cdot \boldsymbol{G} + \frac{\boldsymbol{T}}{\boldsymbol{m}} - \dot{\boldsymbol{o}} \times \dot{\boldsymbol{p}}$$

$$\boldsymbol{J}\ddot{\boldsymbol{o}} = \begin{bmatrix} \boldsymbol{\tau_{roll}} \\ \boldsymbol{\tau_{pitch}} \\ \boldsymbol{\tau_{yaw}} \end{bmatrix} - \dot{\boldsymbol{o}} \times \boldsymbol{J} \cdot \dot{\boldsymbol{o}}$$

donde,

$$\boldsymbol{W^{-1}} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{\frac{s\phi}{c\theta}} & \boldsymbol{\frac{c\phi}{c\theta}} \\ \boldsymbol{0} & \boldsymbol{c\phi} & \boldsymbol{-s\phi} \\ \boldsymbol{1} & \boldsymbol{s\phi \tan\theta} & \boldsymbol{c\phi \tan\theta} \end{bmatrix}$$

Se realiza una hipótesis en la que se supone que todo el empuje generado por los motores se encontrará alineado con el eje z, de forma que el vector T tendrá la forma:

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{T_z} \end{bmatrix}$$

Por lo que la entrada de la planta y salida del controlador que veremos más adelante, será:

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{Tz} \\ \boldsymbol{\tau_{yaw}} \\ \boldsymbol{\tau_{pitch}} \\ \boldsymbol{\tau_{roll}} \end{bmatrix}$$

# 3 PUESTA EN MARCHA Y TOOLBOX MIT

## 3.1 Introducción

La base de este proyecto se encuentra en el conjunto de herramientas que se crearon por los profesores Fabian Riether y Sertac Karaman para una asignatura de sistemas de control, *16.30 Feedback Control Systems*, del Instituto de Tecnología de Massachusetts, *MIT*. El firmware personalizado para el Parrot Rolling Spider permite al software de Parrot llamar a librerías externas en cada iteración, estas librerías escritas en C se encargarán del control completo del drone y pueden ser modificadas según el usuario vea adecuado. Por otro lado, ponen a disposición un Toolbox de *Matlab/Simulink*, este puede simular los sistemas de control y generar el código a partir de los bloques de Simulink, estas herramientas también dejan preparado todo relacionado con la conexión con el drone, subir código modificado, realizar experimentos, descargar datos tomado, etc. De esta forma, se puede pasar directamente al diseño de controladores y aplicaciones en lugar de encontrarse con los problemas relacionados con la parte de telecomunicación con el drone.

Todo lo relacionado con el toolbox del MIT que se explicará a continuación se puede encontrar en: [16] y [17]

## 3.2 Puesta en marcha

A continuación, se detalla los pasos que se deben seguir para poner en funcionamiento un Parrot Rolling Spider recién comprado, desde la instalación del software necesario hasta el primero vuelo del drone.

### 3.2.1 Instalación de software

En primer lugar, destacar que el toolbox se ha diseñado para funcionar en Ubuntu 14.04 por lo que es necesario tenerlo instalado de forma nativa en el disco duro o usar una máquina virtual. En este caso se utiliza VMWare Fusion para emular este sistema operativo. Es necesario también poseer una versión de Matlab que contenga Simulink y para un resultado mejor, el toolbox de procesamiento de imágenes para poder analizar fotografías tomadas desde el drone. Para este proyecto se usará concretamente la versión R2015a.

Señalar que se denomina a la ruta hasta el directorio principal del toolbox como [DIR], en este caso se corresponde a ~/RollingSpiderEdu-master/MIT_MatlabToolbox.

Los programas y librerías que son necesario instalar para el funcionamiento correcto del toolbox son los siguientes:

- Rolling Spider Toolbox, se puede descargar desde [16] y copiar al directorio deseado o usando el terminal:

  ```
  git clone https://github.com/Parrot-Developers/RollingSpiderEdu.git
  ```

- Lftp

  ```
  sudo apt-get install lftp
  ```

- Bluetooth Stack

  ```
  sudo apt-get install bluez-compat
  ```

- Expect

  ```
  sudo apt-get install expect
  ```

- Gcc-arm-toolchain, estos archivos se pueden encontrar en [DIR]/libs/gcc-arm-Toolchain y bastará con descomprimirlo en el directorio principal del disco duro.

- En el caso de usar un sistema operativo de 64 bits necesitaremos algunas librerías adicionales:

  ```
  sudo apt-get install lib32z1 lib32ncurses5 lib32bz2-1.0
  ```

Una vez instaladas las librerías, se debe añadir algunas carpetas a PATH, entorno de variables que permite indicar a la máquina donde buscar los programas que ejecutaremos para poder llamarlos directamente sin tener que indicar su directorio:

```
sudo gedit ~/.profile
```

En el archivo a editar que aparece, se añade al final la siguiente línea, se guarda el archivo y se cierra y se vuelve abrir la sesión para que se efectúe el cambio.

```
export PATH=$PATH:[DIR]/bin:[DIR]/bin/utils:[DIR]/bin/firmware
```

Por último, se debe ejecutar el siguiente comando:

```
BuildUtils.sh
```

### 3.2.2 Instalación de firmware

El Parrot Rolling Spider que se va utilizar en este proyecto tiene instalado de fábrica el firmware comercial con versión 1.99.2, es posible modificar este descargando la versión deseada desde [18] y se puede instalar simplemente copiando el archivo en el directorio principal del drone cuando lo conectas al ordenador y esperando a que dejen de parpadear los LEDs. Esto es necesario tenerlo en cuenta ya que el toolbox está adaptada para esta versión de firmware y es posible que no funcione con futuras actualizaciones.

Para poder usar el drone comercial con el toolbox ya instalado es necesario instalar el firmware personalizado en él. Para ello, se conecta el drone al ordenador con el cable USB que trae, asegurando que se conecta a Ubuntu en el caso de utilizar una máquina virtual y se ejecuta el siguiente comando en una ventana de terminal:

```
EDUfirmwareUploadSYS.sh
```

Se esperan unos 10 segundos hasta que dejen de parpadear las luces delanteras. Una vez se detengan se desconecta el drone del ordenador y se vuelve a conectar, ahora se observa que el directorio del drone contiene un archivo llamado *fvt6.txt*[1] , en su interior hay información general sobre el drone: nombre del producto, dirección MAC, número de serie, versión de firmware, etc.

Se copia la dirección MAC que aparece en el archivo mencionado y se ejecuta el siguiente comando en el

[1] En el caso de no encontrar el archivo fvt6.txt se puede encontrar la dirección MAC del drone con el comando *sudo hcitool lescan,* teniendo el adaptador bluetooth conectado.

terminal para definir la dirección y poder realizar una conexión exitosa más adelante. Es recomendable copiar los datos del drone en un archivo aparte para tenerlos disponible en cualquier momento sin necesidad de tener que conectarlo otra vez.

```
DroneSetMACaddress.sh [DIRECCIÓN MAC]
```

A continuación, se desconecta de nuevo el drone y se le inserta una batería cargada, los LEDs delanteros deberían parpadear múltiples veces y apagarse cuando el firmware se haya instalado correctamente. En el caso de que no lleguen a parpadear, sería conveniente repetir los pasos seguidos.

Una vez instalado, se procede a realizar la primera conexión inalámbrica con el vehículo, para ello se conecta el adaptador bluetooth de bajo consumo y se ejecuta:

```
DroneConnect.sh
```

Es necesario copiar todos los archivos del firmware al drone:

```
EDUfirmwareUploadFILES.sh
```

Este comando sube al drone los archivos que se encuentran en [DIR]/libs/EDUfirmwareFILES/ a través de ftp por la IP 192.168.1.1, como esto se realiza por Bluetooth es necesario desactivar la conexión WIFI para evitar recibir errores, ya que coincide con la dirección del router e intentará conectarse a este dando algún tipo de fallo.

Se reinicia el drone con el comando DroneReboot.sh y se realiza de nuevo una conexión.

En último lugar, queda inicializar el firmware subido y el drone en sí, estos comandos dan los permisos necesarios a los archivos que los soliciten y los mueve a su destino final.

```
EDUfirmwareInitialize.sh
```

```
DroneInitialize.sh
```

Llegados a este punto, se ha subido el firmware personalizado al drone y se han inicializado todos los parámetros. Este proceso solo es necesario realizarlo una vez, a no ser que se resetee el drone de fábrica.

Al igual que se puede establecer una conexión con el drone desde el terminal se puede desconectar con el comando:

```
DroneDisconnect.sh
```

### 3.2.3 Primera simulación

Una vez que el drone tiene instalado el firmware personalizado, ya es el momento de diseñar controladores, simularlos y subirlos al drone para su primer vuelo. Con Matlab iniciado, hay que dirigirse a la dirección [DIR]/trunk/matlab aquí se encuentran 3 carpetas y un archivo:

- **Simulation/**: En ella se encuentran los archivos de Simulink para diseñar y simular el drone.
- **libs**/: Contiene una versión del toolbox de robótica de Peter Corke adaptada al Parrot Rolling Spider. Este toolbox define toda la dinámica del drone para poder realizar una simulación lo más cercana a la realidad. Como se verá más adelante, hay grandes diferencias entre las simulaciones y el vuelo real del drone, ya que existen muchos más factores que intervienen en la dinámica del sistema y complicarían demasiado el modelo.
- **ExperimentAnalyzer/**: Contiene varios archivos para analizar los datos tomados por los sensores y la dinámica del drone, tanto en simulaciones como grabados en vuelo.

Ejecutando el archivo startup.m se añadirán estas tres carpetas a la ruta de búsqueda de Matlab, para poder ejecutar archivos directamente sin necesidad de estar en su carpeta y se desplazará a la carpeta de simulación.

Abriendo el archivo *sim_quadrotor.slx* se iniciará Simulink y mostrará el diagrama de bloques completo del drone.

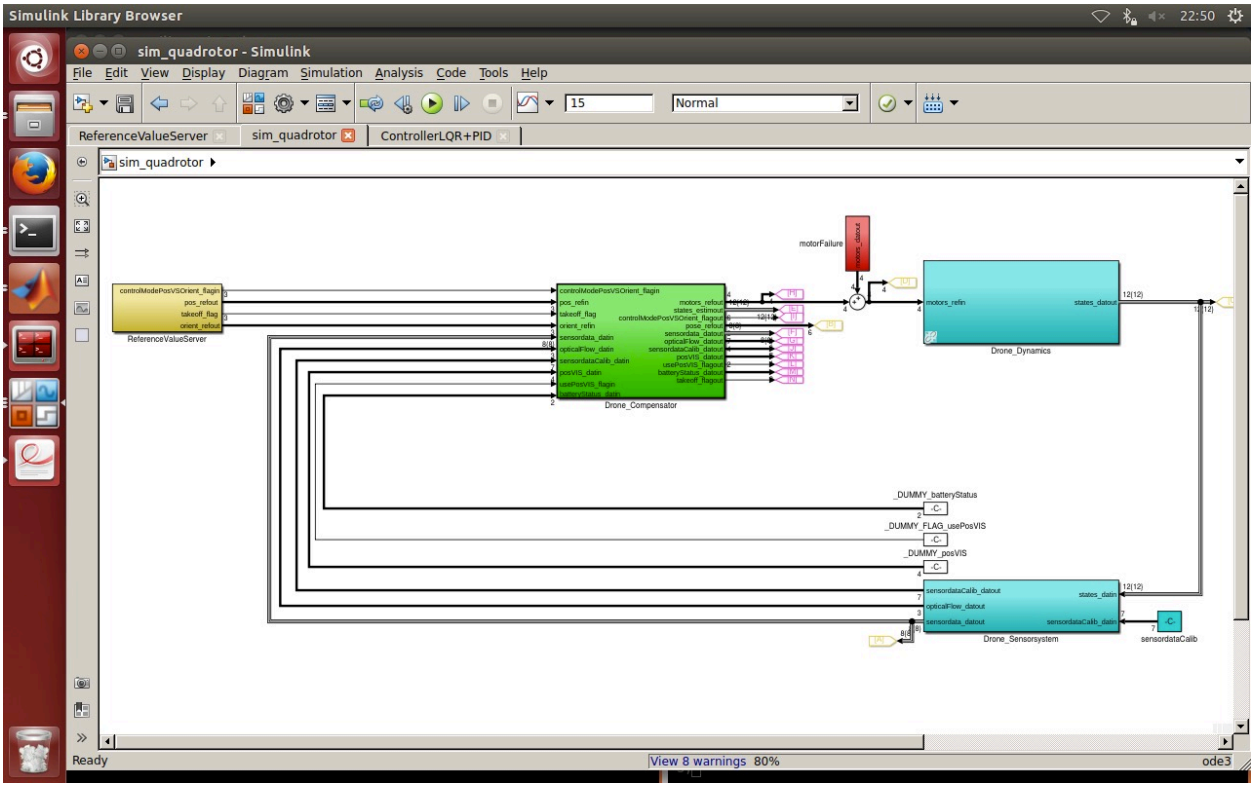



**Figura 3-1 Diagrama de bloques principal**

En la Figura 3-1 se puede ver el diagrama principal, a destacar 4 bloques.

1) **ReferenceValueServer**: Se encarga de dar valores de referencia para la posición y orientación del drone para la simulación. Para el vuelo real del drone será realizado por código como se verá más adelante.

2) **Drone_Compensator**: Se encarga del control del drone, realiza estimaciones de posición y orientación y las compara con los datos de los sensores, obteniendo así un error que será pasado al controlador para intentar corregirlo. Tiene 10 entradas:

   - Bandera de control en posición vs. control en orientación. Este parámetro indica si se realiza un control en posición para mantener una altura indicada o en orientación para realizar variaciones en las referencias del pitch, roll y yaw.
   - Referencia de posición.
   - Bandera de despegue. Activado mientras el drone está despegando, en ese momento el control se centra en llegar a la altura de despegue.
   - Referencia de orientación.
   - Datos de los sensores.
   - Datos del flujo óptico.
   - Datos de la calibración de los sensores.

- Datos de visión tomados por cámara.
- Bandera que indica si se quiere usar los datos de visión.
- Valor de la batería.

Estos serán los parámetros de entrada también cuando se llame a este bloque desde el código para el vuelo del drone. El bloque tiene 11 salidas, muy similares a las entradas, pero modificadas tras pasar por el controlador, incluye también la salida que irá a los motores para compensar el error cometido.

3) **Drone_Dynamics**: Recibe de entrada la salida de los motores y se le aplica al modelo dinámico del drone, a partir del toolbox de Peter Corke, dando como salida las nuevas variables de estado del drone. Como se comentó en el punto anterior, estas variables de estado incluyen la posición XYZ, la orientación en ángulos de Euler, las velocidades lineales y las rotaciones angulares. Será un vector de 12 componentes. Este bloque obtiene la respuesta dinámica del vehículo ante la acción que ejecutarían los motores.
4) **Drone_SensorSystem**: Se encargará de simular los valores de los sensores a partir de las variables de estado del ciclo anterior, mandará sus resultados al bloque de control.

Utilizando el controlador PID que trae por defecto, se puede hacer una simulación para comprobar que todo funciona correctamente. Para ello, se hace click en Run y se espera a que se detenga la simulación. A continuación, a la derecha del diagrama mostrado en simulink se puede observar un conjunto de Scopes que representan todos los datos de la simulación realizada.

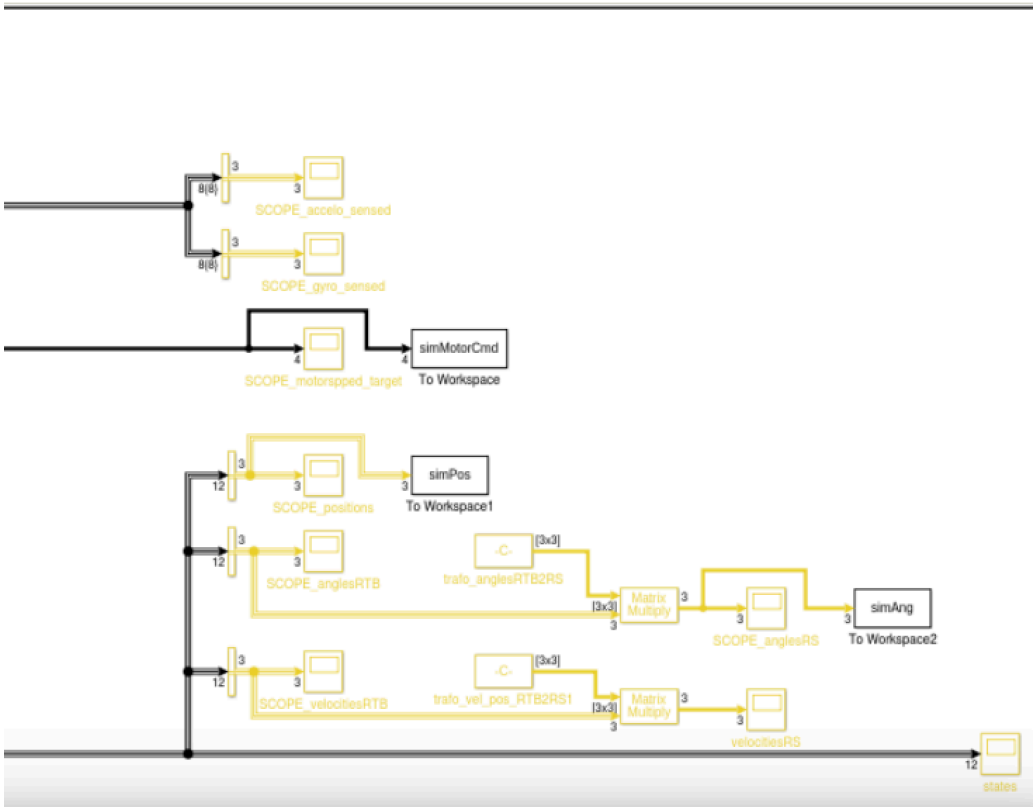



**Figura 3-2 Scopes para la representación de datos simulados**

Por ejemplo, si se desea mirar la posición que ha mantenido el drone a lo largo de la simulación, bastaría con visualizar SCOPE_positions, obteniendo el siguiente gráfico.

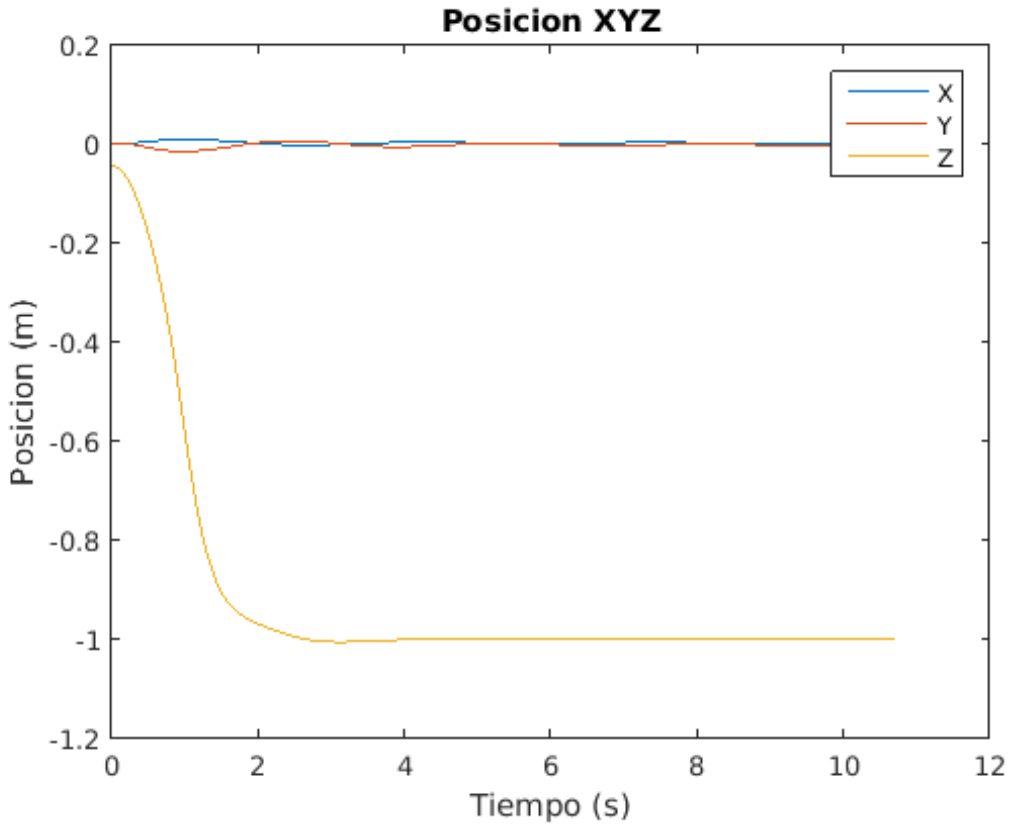



**Figura 3-3 Simulación de la posición**

En esta primera simulación se le indica al drone que mantenga una altura constante de un metro, se observa que realiza un buen trabajo y alcanza la altura deseada en unos dos segundos y la mantiene sin problemas. Como se verá seguidamente, el vuelo del drone real no será tan perfecto con las simulaciones que se realicen.

Para realizar el primer vuelo con el drone es necesario compilar el código, para ello basta con volver al diagrama principal del simulink y haciendo click derecho en el bloque Drone_Compensator seleccionar *Build this Subsystem* dentro de la pestaña *C/C++ Code*. Una vez compilado, ya está listo para subirse al drone, estando conectado al vehículo basta con ejecutar el siguiente comando en una ventana de terminal.

```
DroneUploadEmbeddedCode.sh
```

Por último, es necesario abrir dos ventanas de terminal. En una ejecutaremos DroneKeyboardPilot.sh y en otra

`DroneRun.sh`. El primer comando se encarga de activar el ReferenceValueServer, con el que se podrá controlar manualmente el drone desde el teclado. El segundo manda la orden de despegue, primero realizará una comprobación de que todos los sistemas están correctamente, que se encuentra situado en una posición adecuada y que tiene bastante batería. Una vez realizado, despegará e intentará llegar a la altura deseada y mantenerla. El resultado obtenido con el controlador PID por defecto es el de la Figura 3-4, se observan las mediciones del sensor de presión y del ultrasónico, así como la estimación obtenida del filtro de Kalman y la referencia de altura que se le ordena al vehículo.

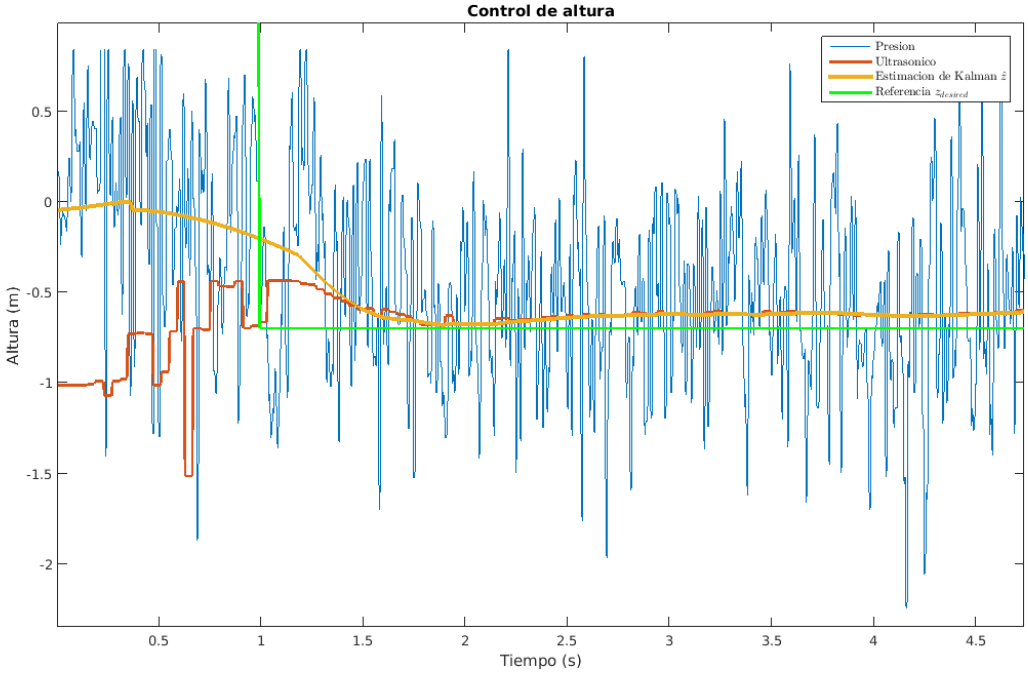



**Figura 3-4 Despegue real con controlador PID**

En este caso, la altura deseada es de 0.70 metros. Se observa como el resultado no es tan perfecto como indicaba la simulación, esto es debido a que en el vuelo real intervienen muchos más factores de los que se tienen en cuenta en el modelo dinámico, aunque para tener una aproximación si será útil.

## 3.3 Desarrollo del Toolbox

En este apartado se desarrollará el funcionamiento de los bloques más significativos del archivo de Simulink para entender su funcionamiento con más detalle, al mismo tiempo se explicarán los archivos de código que entran en juego a la hora de simulación y del vuelo real. Se observará los parámetros que se pueden modificar para cambiar el funcionamiento del drone y las zonas del código más curiosas para añadir funcionalidades al programa.

### 3.3.1 Diagramas de bloques

Como ya se ha mencionado en el punto anterior, en el diagrama inicial se observan cuatro bloques. El bloque situado a la izquierda, *ReferenceValueServer*, es el encargado de enviar órdenes de movimientos a la simulación, desde ahí se pueden crear escalones del tamaño y duración deseados para el roll, pitch o yaw y así poder ver como afectarían esos cambios al controlador diseñado. Del mismo modo es posible modificar la altura de despegue deseada, así como el tiempo de despegue. Siempre es aconsejable realizar una simulación lo más parecida a lo que se desea en la realidad para ver una aproximación de cómo reaccionará el drone ante los cambios de referencia.

El siguiente bloque, *Drone_Compensator*, es el único que se utiliza por el drone para el vuelo real. Entrando en él se puede observar, Figura 3-5, el bloque de estimación, que recoge todos los datos recibidos por los sensores y la cámara para realizar la estimación de las variables de estado y mandárselas al bloque de control, y el bloque del controlador.

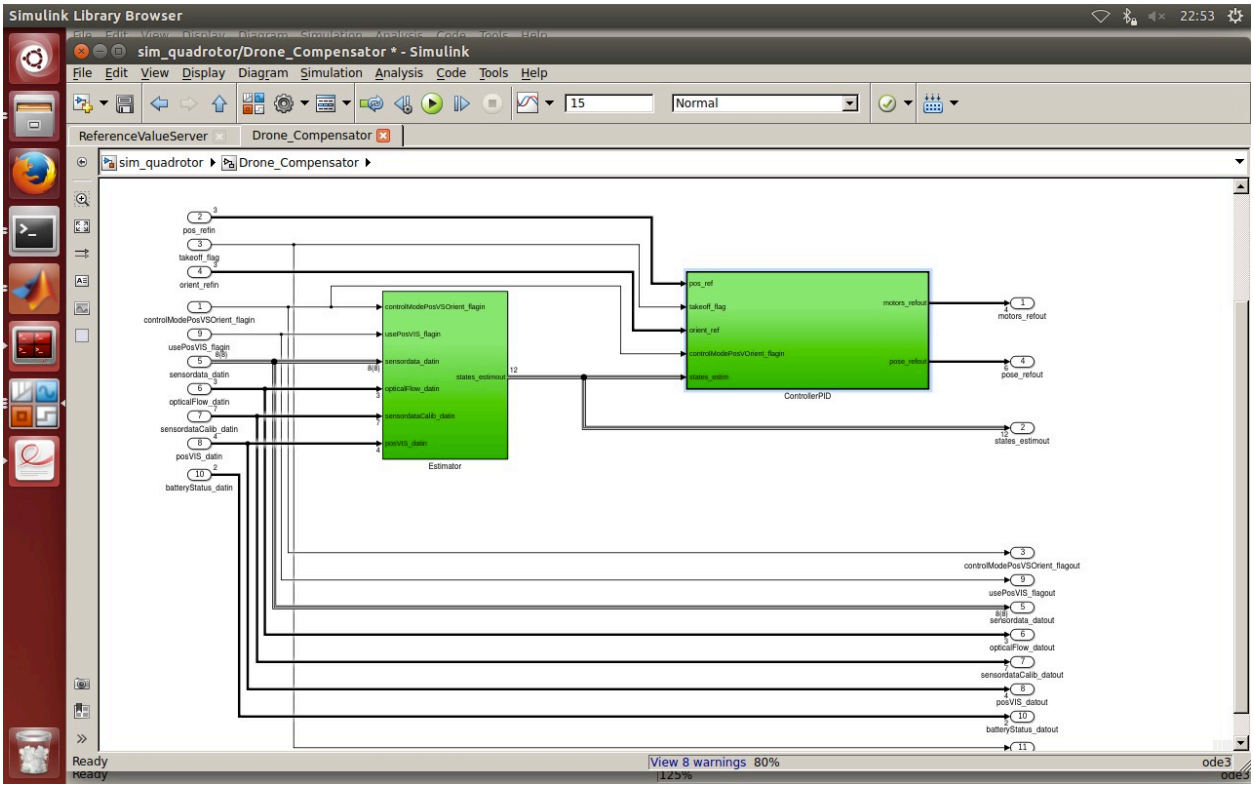



**Figura 3-5 Diagrama de bloques Drone_Compensator**

El bloque de estimación contiene en su interior otros cuatro bloques dentro, su objetivo es conseguir el vector de estados de ese instante para mandárselo al controlador. En primer lugar, el procesamiento de sensores recibe todos los datos de los sensores y los separa de forma que se pueden utilizar a continuación de forma directa. Las salidas son los valores de entrada más ordenados, quitando aquellos que no se utilizarán. Las salidas de este bloque van a otros dos, el primero se encarga de calcular la estimación de los ángulos yaw, pitch, roll y las rotaciones angulares p, q, r, el segundo obtiene la altura del drone y la velocidad lineal en el eje z.

Para el cálculo de la altura de vehículo se utilizan los datos del sensor de presión y del sensor ultrasónico para obtener una estimación, para ello se utiliza el filtro de Kalman [19]. Este procedimiento proporciona la mejor estimación del estado de un sistema, suavizando las medidas con ruido. Es muy utilizado actualmente en el ámbito de los robots móviles. El drone calcula la altura en cada momento según dos sensores: ultrasónico y barómetro. En bajas altura el sensor ultrasónico tiene muy poca fiabilidad, mientras que el barómetro tiene un mejor funcionamiento aunque posee mucho ruido, por ello se utiliza un filtro de Kalman, para obtener el mejor resultado en cada momento y sin ruido.

El filtro de Kalman es un conjunto de ecuaciones matemáticas que permiten un cálculo recursivo de la estimación de los estados de un sistema. Todo ello se realiza en dos fases:

- Predicción: Se realiza una predicción del estado y del error de la covarianza en el instante k a partir del estado previo, k-1. El estado previsto se obtiene a partir del modelo dinámico del sistema.

$$\hat{\boldsymbol{x}}^{-}_{\boldsymbol{k}} = \boldsymbol{A}\hat{\boldsymbol{x}}_{\boldsymbol{k-1}} + \boldsymbol{B}\boldsymbol{u}_{\boldsymbol{k}}$$

$$\boldsymbol{P}^{-}_{\boldsymbol{k}} = \boldsymbol{A}\boldsymbol{P}_{\boldsymbol{k-1}}\boldsymbol{A}^{\boldsymbol{T}} + \boldsymbol{Q}$$

Donde,

$$A = \begin{bmatrix} 1 & 0.005 \\ 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0.005 \end{bmatrix}$$

$$Q = 0.0005$$

- Estimación: En esta fase se hace una corrección del estado y del error de la covarianza en el instante k. Para ello es necesario realizar el cálculo de la matriz de ganancia de Kalman que indica la diferencia entre la medida prevista y la medida real.

$$\boldsymbol{K_k = P^-{}_k H^T (H P^-{}_k H^T + R)^{-1}}$$

$$\boldsymbol{\hat{x}_k = \hat{x}^-{}_k + K_k (z_k - H \hat{x}^-{}_k)}$$

$$\boldsymbol{P_k = (I - K_k H)\, P^-{}_k}$$

Donde,

$$H = 0$$

$$R = 0.1$$

A partir de este procedimiento se obtiene una buena aproximación de la altura a la que se encuentra el drone utilizando ambos sensores mencionados. El otro bloque que se encuentra dentro de *Drone_Compensator* es el controlador, este será encargado de recibir el vector de estados estimados e intentar llevar el estado actual hasta el estimado a partir de una orden de comandos hacía los motores. Este bloque se explicará con muchos más detalles en el punto 4, sistemas de control, donde se analizarán distintos controladores y a partir de distintas pruebas se decidirá cuál será el más adecuado para utilizar para desarrollar aplicaciones sobre el vehículo.

Otro bloque de vital importancia para la simulación es *Drone_Dynamics*, este recibe como entrada la salida del controlador utilizado, es decir, la orden que irá a los motores del drone. Aquí se transforma el comando de los motores en velocidades angulares y, seguidamente, se le aplica la librería de robótica de Peter Corke [20]. De esta forma se obtiene un vector de estados con todos los datos del drone, los cuales se enviarán al bloque encargado de simular los sensores del vehículo. Este conjunto de valores indica cómo reaccionaría el drone a la orden enviada a los motores que, como se ha visto, no es exactamente fiel a la realidad.

Por último, se encuentra el bloque *Drone_Sensorsystem*, encargado de simular el resultado de los sensores del UAV partiendo del vector de estados obtenido en el bloque *Drone_Dynamics*. En este bloque se consigue datos de la unidad de medición inercial, *IMU*, que informará acerca de aceleración, velocidades y fuerzas gravitatorias.

### 3.3.2 Detalles del Código

Todo el código mencionado en este apartado se podrá encontrar en la carpeta "Archivos Importantes" del repositorio de Github [2]. Señalar que la base del código sigue estando en [16] aunque se han realizado numerosos cambios para obtener unos resultados diferentes al original.

Se comienza con el código principal, el encargado del control del drone, ***rsedu_control.c***. Este archivo

programado en lenguaje C incluye todo lo equivalente a los bloques *Drone_Dynamics* y *Drone_Sensorsystem* del archivo simulink, es decir, lee todos los datos recibidos por los sensores y los introduce en variables que se envían al bloque *Drone_Compensator* para que se estime el vector de estados actual y se genere una orden de control para intentar llegar al estado deseado.

En las siguientes tablas se puede observar un conjunto de variables aptas para ser modificadas para conseguir cambios en ciertos aspectos del vuelo real.

**Tabla 2 Variables de duración**

| Variable | Utilidad |
|---|---|
| **onCycles** | Números de ciclos que el drone permanecerá encendido. |
| **calibCycles** | Números de ciclos usados para la calibración inicial. |
| **takeoffCycles** | Números de ciclos usados en el despegue del drone. |

**Tabla 3 Características especiales**

| Característica | Utilidad |
|---|---|
| **FEAT_TIME** | Activa el procesamiento de tiempo, guarda el tiempo que tarda en entrar y salir de funciones. En este proyecto 0. |
| **FEAT_OF_ACTIVE** | Añade al código de control los datos recibidos por el flujo óptico para la estimación de velocidad. En este proyecto 1. |
| **FEAT_POSVIS_RUN** | Activa el procesamiento de imágenes, permite el análisis por píxel de la fotografía tomada. En este proyecto 1. |
| **FEAT_POSVIS_USE** | Incluye los resultados de visión para la estimación de la posición. En este proyecto 0. |
| **FEAT_IMSAVE** | Registro de las imágenes tomadas. Si se pone a 1 guardará todas las imágenes tomadas y si se inicializa a 2 activará la función de streaming. En este proyecto 0. |
| **FEAT_NOSAFETY** | Si se inicializa a 1 el drone no se apagará automáticamente al detectar un choque, es más peligroso, pero permite realizar maniobras más acrobáticas. En este proyecto 0. |

Estas características se leerán por defecto desde un archivo situado dentro del drone, el valor inicializado en este archivo C se usará en el caso de no encontrase el mencionado. Para modificar ese archivo basta con

conectarse al drone y utilizar el comando telnet 192.168.1.1 para entrar dentro del directorio del drone. Una vez ahí, modificamos el archivo deseado con vim, vi /data/edu/params/paramsEDU.dat.

La función principal de este archivo es *void RSEDU_control()*, esta recibe los datos enviados por los sensores y genera los comandos de referencia de los motores. Esta porción de código se llamará con una frecuencia de 200*Hz*, es decir, 200 veces por segundo o una vez cada 0.005 segundos.

El vuelo del UAV puede definirse en 4 fases:

- Fase 1: **Inicialización, conexión a servidor y lectura de parámetros**. Durante los dos primeros ciclos se procede a comprobar si existe el archivo *paramsEDU.dat* para inicializar las características que usará el drone. Seguidamente se conectará al servidor encargado del control manual, *ReferenceValueServer*, en el puerto 12345. Más adelante en este apartado se explicará el funcionamiento del control manual. Por último, se activa los servicios deseados según las características que estén activas y se inicializan los sensores.
- Fase 2: **Calibración**. Esta fase tendrá lugar hasta que el contador de ciclos llegue al valor *calibCycles*. Se procede a realizar mediciones de los distintos sensores durante el tiempo de la fase, obteniendo una media de cada valor. De esta forma, se conocen los valores iniciales a los que se encuentra el drone cuando va a despegar.
- Fase 3: **Inicialización de modelo dinámico**. Ocurrirá durante un único ciclo después de acabar la calibración. Aquí se comprueba que el drone se encuentra en una superficie nivelada para el despegue y que los servicios activados están en funcionamiento correctamente. Además, se inicializa el modelo dinámico del drone, es decir, las entradas y salidas correspondientes al bloque *Drone_Compensator* de Simulink.
- Fase 4: **Vuelo**. Esta última fase tendrá lugar hasta que pasen *onCycles* ciclos. A su vez, esta fase está dividida en 3 zonas:
    - Despegue: ocurrirá mientras no pasen más de *calibCycles* + *takeoffCycles*. El drone despegará durante el tiempo indicado.
    - Transición a vuelo: Una vez pase el tiempo de despegue, se activará el control de altura a la deseada.
    - Vuelo real: El drone ya está de camino a la altura deseada utilizando el controlador diseñado. En esta zona ya se está realizando un control del drone y se encuentra a la espera para recibir comandos a través del socket. También estará atento por si sucede algo que requiera una parada de emergencia, por ejemplo, que haya bajado la batería por debajo del límite establecido o que haya detectado un choque por un cambio brusco en una de las aceleraciones. Del mismo modo, irá actualizando el vector de estado que recibe el bloque de simulink para poder seguir controlando hacía el estado deseado.

Para terminar el análisis de este archivo se mencionan las variables más destacables utilizadas en la función de control durante el vuelo (Tabla 4).

**Tabla 4 Variables destacables**

| Variable | Utilidad |
|---|---|
| **counter** | Lleva la cuenta de los ciclos pasados. |
| **MIN_BATTTAKEOFF / MIN_BATT** | Mínimo de batería para poder realizar un despegue. / Mínimo de batería para volar. |
| **MAX_ACCELL** | Aceleración a partir de la cual se reconoce un choque. |
| **sensorcal [7]** | Vector de 7 componentes que se irá actualizando con los valores de aceleraciones xyz, medidas del giroscopio en xyz y la medida del sensor de presión. |
| **recBuff [100]** | Buffer en el que almacenará la información recibida por el socket. |

El siguiente archivo que se analizará es ***ReferenceValueServer.c***. Este código será el encargado del control manual del drone desde el ordenador. Se ejecuta al llamar al comando *DroneKeyboardPilot.sh* desde una ventana de terminal. Al iniciarse el programa se creará un socket TCP en escucha en el puerto 12345, el cual recibirá una conexión al mandar la orden de despegue con el comando *DroneRun.sh* en otra ventana de terminal. Un socket es, básicamente, una interconexión entre procesos. El protocolo TCP está orientado a conexión y aporta una gran confiabilidad, asegurando la llegada del mensaje enviado. En el caso de que no se reciba el mensaje, la parte del cliente volverá a enviarlo, hasta que reciba la respuesta indicando que ha sido recibido [21].

Una vez se ha recibido la petición de conexión y se ha aceptado, se pueden enviar comandos por el socket. Estos comandos se enviarán siguiendo siempre el mismo formato, como se puede observar en la tabla 5. De esta forma, siempre que haya una modificación en la referencia que cualquier eje que se quiere controlar será necesario enviar la traza completa del mensaje, esta se recibirá y se almacenará en la variable *recBuff* de rsedu_control.c y serán reconocidos e introducidos en el modelo.

**Tabla 5 Comando enviado por socket**

| Runcmd | Pitch | Roll | Yaw | Altura | Aterrizaje |
|---|---|---|---|---|---|
| Entero | Entero | Entero | Entero | Entero | Entero |

Dentro de esta cadena cabe señalar que la variable *runcmd* se utiliza para parar por completo el drone y por otro lado, la variable *parar_flag* se utilizará para ordenar un aterrizaje progresivo.

Esta traza se enviará cada vez que se pulse una tecla que realice alguna modificación sobre el estado actual del drone, en la tabla 6 se puede observar los distintos comandos posibles y su intención. Del mismo modo, en la Figura 3-6, se observa un esquema del UAV indicando los ejes de Pitch, Roll y Yaw, para indicar el signo.

**Tabla 6 Comandos para control manual**

| Comando | Resultado |
|---|---|
| **D** | Roll += 0.04 |
| **A** | Roll −= 0.04 |
| **W** | Pitch −= 0.04 |
| **S** | Pitch += 0.04 |
| **I** | Altura −= 0.2 |
| **K** | Altura += 0.2 |
| **J** | Yaw −= 0.2 |
| **L** | Yaw += 0.2 |
| **E** | Runcmd = 0 |
| **Z** | Aterrizaje_flag = 1 |

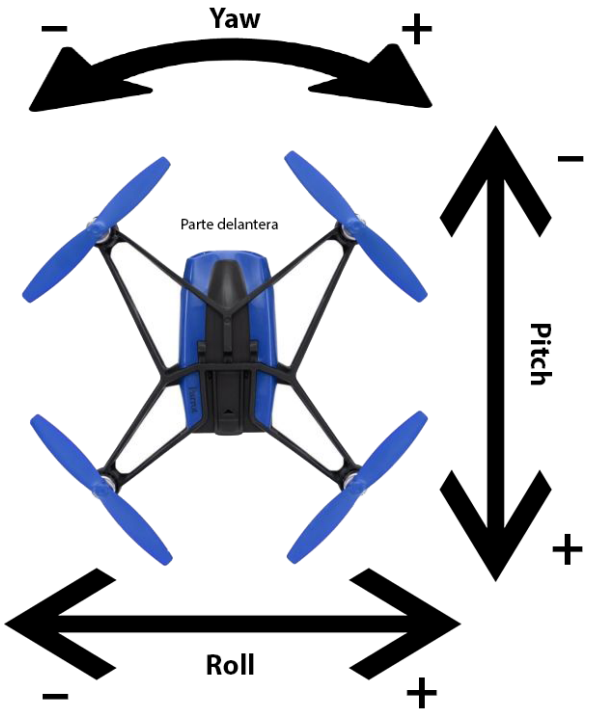



**Figura 3-6 Esquema de drone con direcciones**

Por último, señalar que, antes de enviar los comandos, se comprueba si su valor se encuentra por encima del valor máximo/mínimo establecido para proceder a saturarlos y que así no supere estos parámetros de seguridad.

A continuación, se explicará el funcionamiento del código encargado de todo lo relacionado con la cámara, desde el análisis de imágenes hasta el streaming, ***rsedu_vis.c***. La función principal de este archivo es *RSEDU_image_processing,* en ella se desarrolla la mayor parte del código del programa y será lo que se desarrolla seguidamente. Esta parte del programa se llama con una frecuencia de 60*Hz,* 60 veces por minuto, y recibe como entrada un puntero a la imagen que está viendo la cámara vertical del drone en ese instante. Esta fotografía tiene un tamaño de 160x120 píxeles y se encuentra en formato YUV, aunque se utilizará una variable de tipo *pixel2_t* que tendrá un tamaño de 80x120.

El formato YUV consiste en una separación del brillo y del color de forma que se reduce la información de los colores sin permitir que el ojo humano lo llegue a notar. Este espacio de colores se encuentra ampliamente usado en los sistemas de color PAL y NTSC en las televisiones. Está compuesto por una señal luma, Y, que contiene la información de luminosidad de la imagen, equivalente a una versión en blanco y negro de la imagen, y dos componentes de crominancia, UV, que transporta la información del color de la fotografía. Esto es bastante diferente al espacio de color RGB que contiene tres canales de colores, rojo, verde y azul. Normalmente, se utiliza el formato YUV frente al RGB debido a que el ojo humano tiene una mayor sensibilidad hacía el color que verde que a los otros dos y con este cambio se obtiene una escala más uniforme. Al mismo tiempo, el espacio de color YUV permite que la imagen ocupe menos espacio y no tenga pérdidas, lo que la convierte en ideal para la transmisión y el tratamiento de imágenes [22].

Para las imágenes, como se ha mencionado anteriormente, se utiliza el formato *pixel2_t*, que define a una pareja de píxeles consecutivos, un macropíxel, detallado por cuatro *bytes*, dos Y y UV. Esto está creado en forma de estructura en la cabecera del archivo C.

Entrando en el código de la función mencionada, en primer lugar, se realiza la conexión necesaria con el programa de control para poder realizar el envío de información entre ambos archivos. Si se tiene el parámetro *FEAT_IMSAVE* inicializado a 1 entonces se procede a crear una carpeta temporal donde se irán guardando las imágenes en formato *.bin'* tomadas a una velocidad de 10 por segundo. Estas se encuentran dentro del drone, por lo que para recibirlas en el ordenador es necesario abrir una ventada de comandos y ejecutar *DroneDownloadImages.sh* y una vez descargadas, hay que procesarlas con el comando *VisionPrePostProcessor.sh* que creará las imágenes en formato visible.

El análisis de imágenes se utiliza en este proyecto para la detección de una plataforma de aterrizaje que indica al vehículo donde pararse, estabilizarse y bajar. La velocidad de análisis es de cuatro imágenes por segundo y el procedimiento consiste en un recorrido píxel a píxel de la matriz de la imagen, se sitúa en una fila y analiza todas las columnas de ella y al terminar, pasa a la siguiente fila. El enfoque tomado para la detección de la plataforma se desarrollará en el punto 5 de aplicaciones.

Para terminar el análisis del archivo *rsedu_vis.c*, se detallará los pasos a seguir para conseguir una retransmisión de video en directo o *streaming* desde el drone. El funcionamiento de este modo es el siguiente: en cada iteración se guarda la imagen obtenida en una estructura de datos *FIFO, First In-First Out*, y seguidamente, se envía dicha fotografía a un ordenador remoto con Ubuntu instalado a partir de unos comandos.

Como ya se ha mencionado, es necesario inicializar el parámetro *FEAT_IMSAVE* a 2. Serán necesarias cinco ventanas de comandos para conseguir el funcionamiento deseado, dos de ellas se utilizarán para ejecutar el vuelo normal del vehículo.

1. Primero, es necesario conectarse al drone por bluetooth, *DroneConnect.sh*, y entrar en el directorio del vehículo, *telnet 192.168.1.1*. Una vez conectados, se desea recoger cada imagen que guarda el drone y mandarla por un socket a la dirección y puerto deseado, para ello se utiliza el siguiente comando:

   ```
   while [ 1 ]; do cat /tmp/picture | nc 192.168.1.2 1234; done
   ```

   Al ejecutarlo salen mensajes indicando que no es posible conectarse a dicha dirección, pero una vez se

ejecuten todos los comandos, funcionará sin problemas.

2. El siguiente comando se ejecutarán desde un terminal sin estar dentro del directorio del drone. Este creará una estructura FIFO donde introducirá cada imagen recibida por el puerto socket, es decir, esta ventana será el cliente que solicita al servidor la imagen.

   mkfifo /tmp/rollingspiderpicture ; while [ 1 ]; do nc -l 1234 > /tmp/rollingspiderpicture; done

3. El último comando es el encargado de representar la imagen recibida en una ventana del reproductor multimedia mplayer. En él, se especifica el formato y tamaño de la imagen que se va a recibir, así como el directorio donde se podrá encontrar.

   ```
   mplayer -demuxer rawvideo -rawvideo w=160:h=120:format=yuy2 -loop 0 /tmp/rollingspiderpicture
   ```

De esta forma, se obtiene una imagen en streaming como se puede observar en la Figura 3-7. Las ventanas numeradas corresponden con los pasos seguidos ordenadamente. La primera pertenece al comando *DroneKeyboardPilot.sh* y la segunda a *DroneRun.sh*. El resto siguen el orden de los puntos explicados anteriormente.

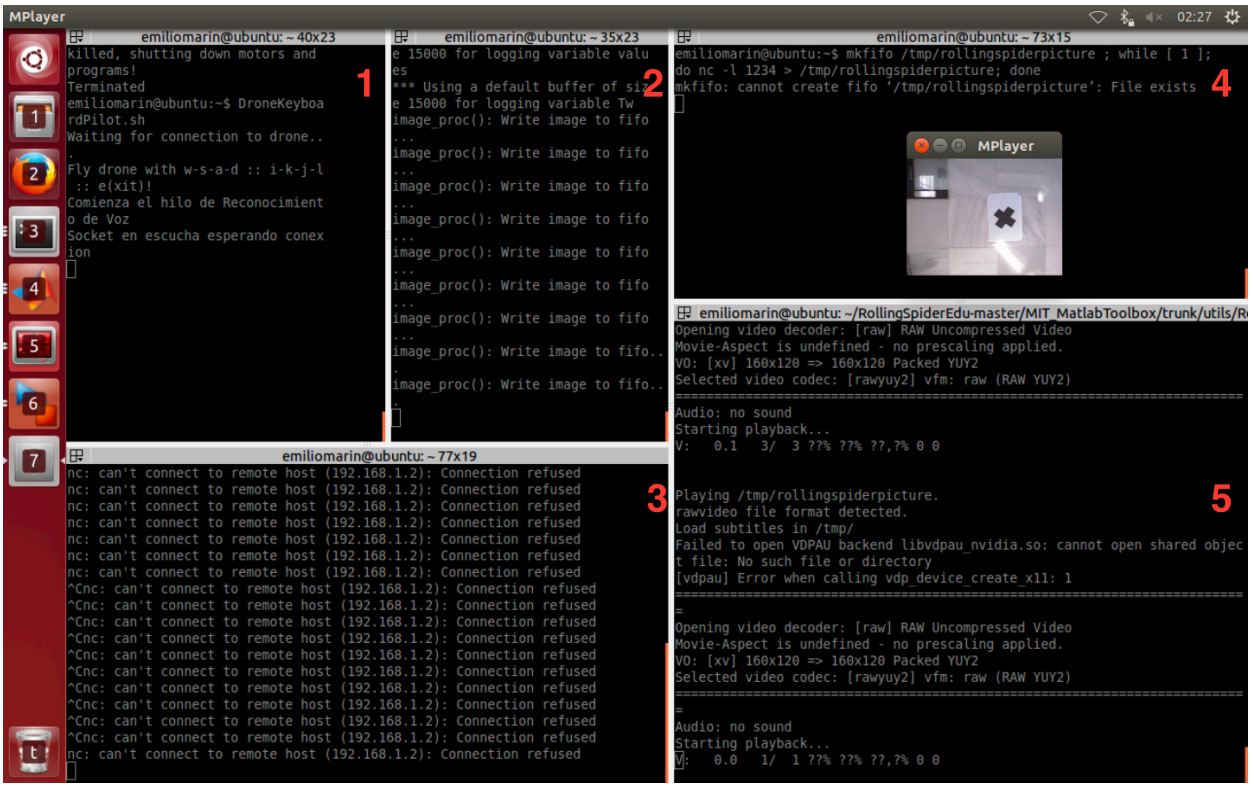



**Figura 3-7 Obtención de streaming**

Para finalizar este apartado, se analiza el archivo que se encarga de la dinámica del quadrotor, ***mdl_quadrotor.m***. Este se encuentra definido dentro de la librería de robótica de Peter Corke [20] y se utilizará para poseer una mayor información a la hora de diseñar controladores y realizar simulaciones en *Simulink*. A diferencia de los códigos analizados anteriormente, este está escrito en lenguaje *MATLAB*. Al ejecutarse crea tres variables dentro del espacio de trabajo de Matlab, tabla 7. Esta librería contiene todas las variables dinámicas necesarias para el desarrollo de un drone y es posible modificar parámetros como: el número de rotores del vehículo, su masa, ganancias, etc. Para este Toolbox en concreto, se ha modificado con los valores

precisos del Parrot Rolling Spider.

**Tabla 7 Variables dinámicas del quadrotor**

| Variable | Utilidad |
|---|---|
| **quad** | Contiene información acerca de las variables concretas del vehículo utilizado, dimensiones, pesos, número de motores, etc. |
| **quadEDT** | Este parámetro está enfocado hacía los sensores y la simulación. Provee ganancias y saturaciones que serán utilizadas por el código al recoger la información de los sensores y analizarla. |
| **sampleTime_qcsim** | Tiempo de muestreo. |

# 4 SISTEMAS DE CONTROL

En este siguiente apartado se realizará el diseño del sistema de control de vehículo. El objetivo de estos controladores será el de corregir los distintos valores vistos (roll, pitch, yaw, altura) de forma que se vayan adaptando a los requerimientos que se le propongan. Por ejemplo, en el momento en el que se le da una orden de avanzar, se disminuye el pitch, el controlador detecta este cambio en forma de un error calculado en ese instante y procederá a dar la orden correcta a los motores para conseguir el vector de estado deseado.

Se diseñarán tres controladores y se recrearán una serie de escenarios, experimentos, para observar cómo responden cada uno de estos sistemas. Esto se hará tanto con simulaciones como utilizando el vuelo real del drone, por lo que se podrá ver también la gran diferencia que hay entre ambos, ya que el vuelo real no es tan ideal como lo puede ser la simulación, entran en juego muchos más factores que no se contemplan por la simulación.

Estos controladores se diseñarán en contínuo ya que el curso seguido del MIT, encargado del desarrollo de esta plataforma, lo explica de esta forma, aunque internamente debe existir una discretización para realizar el control a partir de un tiempo de muestreo. Siendo este tiempo pequeño, los resultados de un controlador contínuo y otro discreto son prácticamente iguales.

Como cabe esperar, únicamente se probarán sistemas de control en lazo cerrado realimentados, aquellos que la salida del sistema afecta directamente sobre la salida de control. Dada una referencia se calcula el error con el estado actual, el cuál va modificando en cada tiempo de muestreo, por lo que este error se ve alterado en cada iteración, variando así la respuesta del controlador.

Los controladores escogidos son: PID, LQR y por asignación de polos (*Pole Placement)*. Se han elegido debido a los conocimientos adquiridos durante las clases de control asistidas en la universidad. El sistema de control óptimo, el que proporcione mejores resultados, será el que se utilice para el apartado de aplicaciones.

## 4.1 Controlador PID

Como ya se ha mencionado anteriormente, el Toolbox utilizado trae por defecto un controlador PID, en realidad múltiples controladores como se verá a continuación. No cabe extrañar el uso de este controlador para un sistema tan complejo como el de un quadrotor, de hecho, este sistema de control es uno de los más utilizados en la industria debido a su sencillez y buen comportamiento. En este apartado se desarrolla la teoría detrás de este controlador y las formas de sintonización de los parámetros del mismo. Ya que, una vez se consigue un buen ajuste de los parámetros del controlador, se logra un rendimiento excelente.

Según la teoría [23] y [24], la ecuación temporal de un controlador PID, que se corresponde con el diagrama de la Figura 4-1, es:

$$\boldsymbol{u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(t)\,dt + T_d\frac{e(t)}{dt}\right)}$$

donde u(t) es la señal de control y e(t) es el error. La señal deseada se obtiene a partir de tres términos, P (proporcional), I (integral) y D (derivativo), cada uno ayuda a solventar ciertos problemas del error. Se observan también tres parámetros asociados a cada término de la ecuación: la ganancia proporcional K, el tiempo integral Ti y el tiempo derivativo Td.

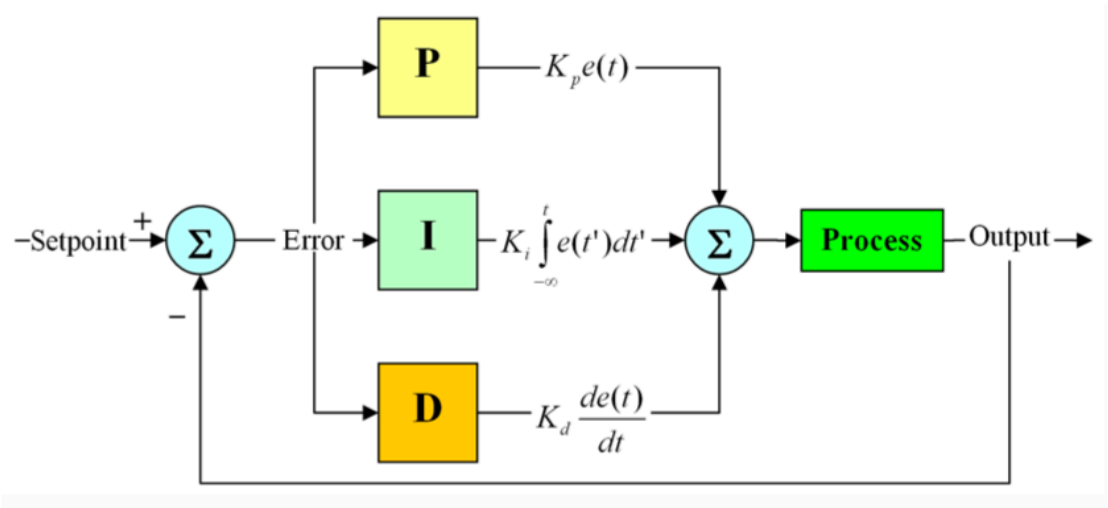



**Figura 4-1 Diagrama de bloques de un PID teórico** [23]

- Término proporcional: Se obtiene una salida proporcional al error de ese instante.

$$\boldsymbol{u(t) = Ke(t)}$$

  En el caso de no requerir un control exhaustivo ante sobreoscilaciones o errores en régimen permanente, un simple controlador proporcional sería el adecuado para controlar el sistema.

- Término integral: El objetivo es la mejora del error en régimen permanente o estacionario, intenta hacer que coincida la salida del controlador con el punto de referencia tomado. Si el parámetro de este término, el tiempo de acción integral, es muy pequeño, la salida nunca llegará a situarse cerca de la referencia deseada en el régimen permanente, mientras que, si es muy grande, se generará una oscilación sobre el valor deseado.

- Término derivativo: Este se utiliza como mejora del régimen transitorio, corrigiendo también la estabilidad en lazo cerrado. Intenta predecir el valor del error futuro pasado un tiempo Td. Si este parámetro es muy pequeño, se obtendrán reacciones lentas ante un cambio de referencia; si es muy grande se conseguiría una respuesta brusca con sobreoscilación, lo que dificultará el control de la señal.

La elección de los parámetros del PID conlleva un compromiso entre el error permitido y el tiempo de subida permisible. Este controlador no obliga a utilizar todos los términos en el sistema de control, se pueden utilizar variaciones del PID como, por ejemplo, el controlador PI o PD, dependiendo del resultado que se desee conseguir. Como se verá, el sistema de control diseñado para este vehículo posee cuatro controladores PID o variaciones del mismo.

El ajuste de los parámetros del controlador suele ser una tarea experimental más que teórica. Existen una serie de pasos aconsejables a seguir cuando se intenta ajustar estos valores de forma práctica:

1. Acción proporcional: Se inicia el tiempo integral Ti en su máximo valor y el tiempo derivativo Td en su mínimo valor, anulando así estos dos términos. Comenzando con una ganancia K pequeña, se va aumentando hasta obtener una pequeña oscilación frente a un cambio de referencia.
2. Acción integral: Se comienza a disminuir Ti hasta que el error en régimen permanente sea anulado, aunque la oscilación en el régimen transitorio sea elevada.

3. Acción derivativa: Manteniendo los valores anteriores, se va aumentando Td hasta obtener una respuesta rápida sin elevada sobreoscilación.

A parte de este método comentado, existen otros dos métodos algo más teóricos a partir de las reglas de Ziegler-Nichols.

El primero consiste en observar la respuesta del sistema a una entrada escalón unitario (Figura 4-2).

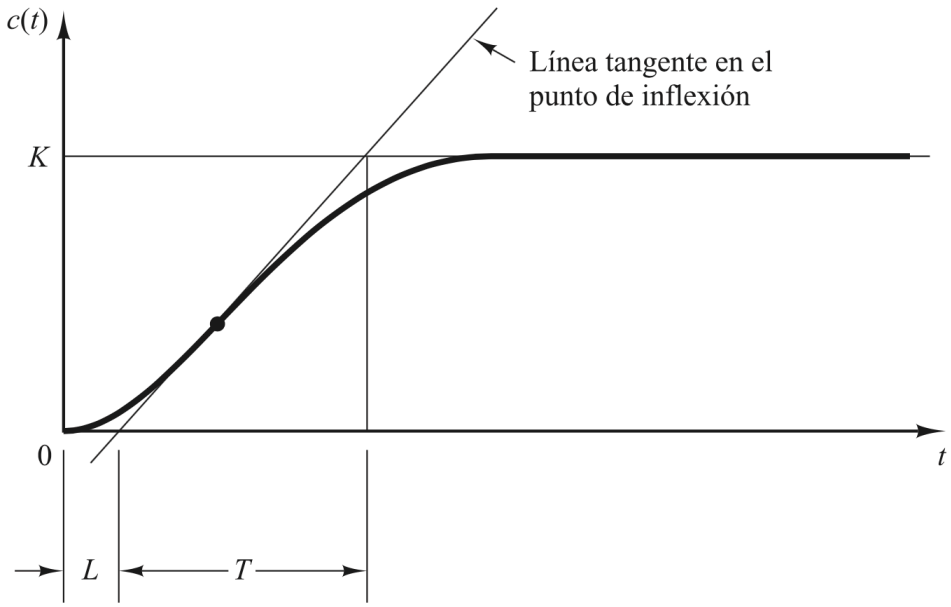



**Figura 4-2 Curva de respuesta ante escalón unitario** [24]

Esta curva está definida por tres parámetros: el tiempo de retardo L, la constante de tiempo T y la ganancia alcanzada K. Con estos parámetros se puede obtener la sintonía del controlador PID según la variante de este que se utilice.

**Tabla 8 Regla de sintonía en función de la respuesta en escalón**

| Controlador | $K_p$ | $T_i$ | $T_d$ |
|---|---|---|---|
| P | $\frac{T}{L}$ | $\infty$ | 0 |
| PI | $0.9\frac{T}{L}$ | $\frac{L}{0.3}$ | 0 |
| PID | $1.2\frac{T}{L}$ | $2L$ | $0.5L$ |

El segundo método teórico se puede considerar similar al primero experimental mencionado. Se parte con los términos integral y derivativo anulados, $T_i = \infty \; y \; T_d = 0$, es decir, un controlador proporcional. En este punto, se va incrementando la ganancia proporcional hasta alcanzar oscilaciones sostenidas, denominando a esa ganancia $K_{cr}$, crítica. Se tendrá una respuesta como la representada en la Figura 4-3, oscilaciones con un periodo constante, crítico, $P_{cr}$, en segundos.

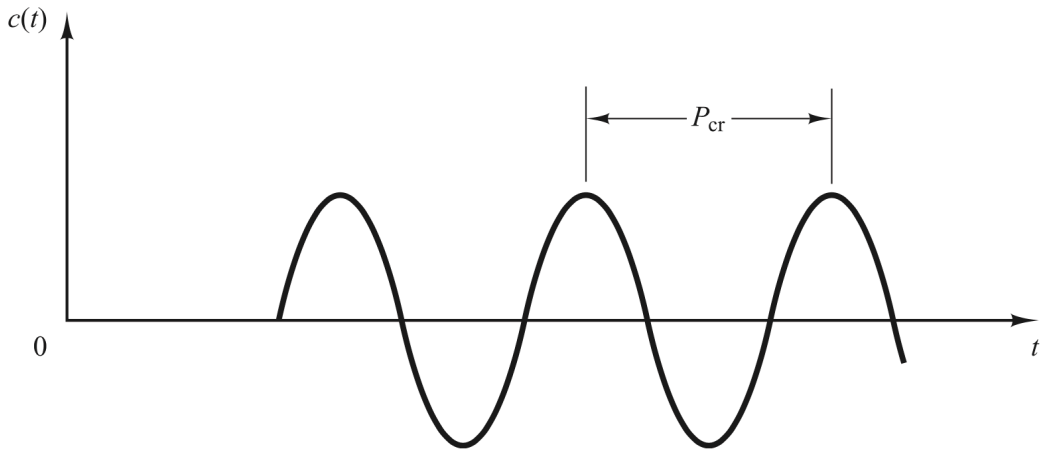



**Figura 4-3 Oscilación sostenida** [24]

Con estos dos parámetros mencionados se obtiene una tabla similar a la mostrada en el primer método teórico.

**Tabla 9 Regla de sintonía en función ganancia y periodo crítico**

| Controlador | $K_p$ | $T_i$ | $T_d$ |
|---|---|---|---|
| P | $0.5K_{cr}$ | $\infty$ | 0 |
| PI | $0.45K_{cr}$ | $\frac{1}{1.2}P_{cr}$ | 0 |
| PID | $0.6K_{cr}$ | $0.5P_{cr}$ | $0.125P_{cr}$ |

Aun así, estos métodos pueden parecer bastante complejos ya que no se tiene una única salida a controlar sino múltiples estados controlables.

Se ha desarrollado la teoría de este controlador de forma continua ya que es como se explica en el curso del *MIT* seguido, aunque también podría razonarse desde un punto de vista discreto, ya que no hay duda de que al estar utilizando un control por computador este no será completamente continuo. Existen varias formas de discretización de este controlador [25], por ejemplo la aproximación bilineal que tiene el mejor funcionamiento de las estudiadas.

La derivada se puede aproximar por:

$$\frac{\boldsymbol{de(t)}}{\boldsymbol{dt}} = \frac{\boldsymbol{e_k - e_{k-1}}}{\boldsymbol{T}}$$

donde T es el tiempo de muestreo.

El término integral, se aproxima como:

$$\int_0^t \boldsymbol{e(\tau)d\tau} = \sum_{i=1}^{k} \boldsymbol{T}\frac{\boldsymbol{e_i + e_{i-1}}}{\boldsymbol{2}}$$

Por lo que se llega a una ecuación discretizada del controlador:

$$u_k = K\left(e_k + \frac{T}{T_i}\sum_{i=1}^{k}\frac{e_i + e_{i-1}}{2} + \frac{T_d}{T}(e_k - e_{k-1})\right)$$

A pesar de estas diferencias, para tiempos de muestreo muy bajos el funcionamiento de la ecuación discretizada y la temporal es muy parecido.

Como ya se ha mencionado, se tiene una combinación de controladores, uno por cada sistema que se quiere controlar. En concreto, uno para la posición X-Y, otro para el yaw, otro para roll y pitch y por último, uno más para el control de altura. En el diagrama de bloques, el bloque del controlador recibe como entrada cuatro parámetros: referencia de posición, referencia de yaw, pitch y roll, control de posición frente a control de ángulos y la estimación del vector de estado.

La Figura 4-5 muestra el sistema de control final.

Analizando cada subsistema por separado:

- Posición X-Y: Este controlador se utilizará únicamente cuando la bandera que señala el control en posición frente al control de ángulos sea mayor que cero. En ese caso, entrará en juego este controlador PD. Este recibe la posición XY actual, su estimación y la estimación de las velocidades en los mismos ejes. Como se observa, se obtiene el error entre la posición actual y la estimación y le aplica el término proporcional, mientras que a las velocidades le aplica el término derivativo y estos dos son sumados, calculando así el comando de pitch y roll que se le enviará a su controlador.

  En el caso que la bandera sea igual a cero, el comando de pitch y roll que se enviará será el de la referencia de ángulos directamente.

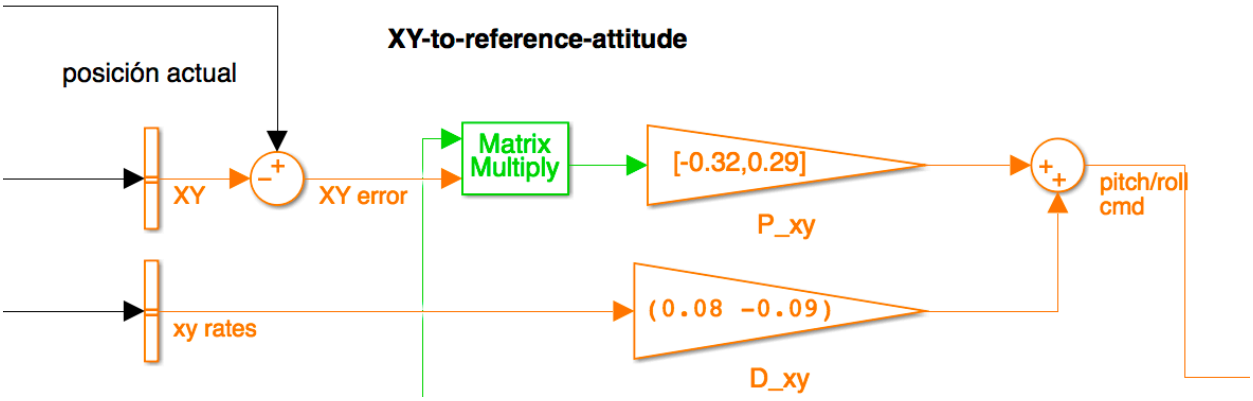



**Figura 4-4 Controlador PD para posición XY**

- Yaw: Este controlador calcula el error cometido en el yaw a partir de la diferencia entre la referencia de ángulos y la componente del vector de estados y de forma similar al anterior, le aplica un término proporcional. Mientras que a la rotación angular en el eje z se le aplica el derivativo. Su salida va directamente al *ControlMixer*.

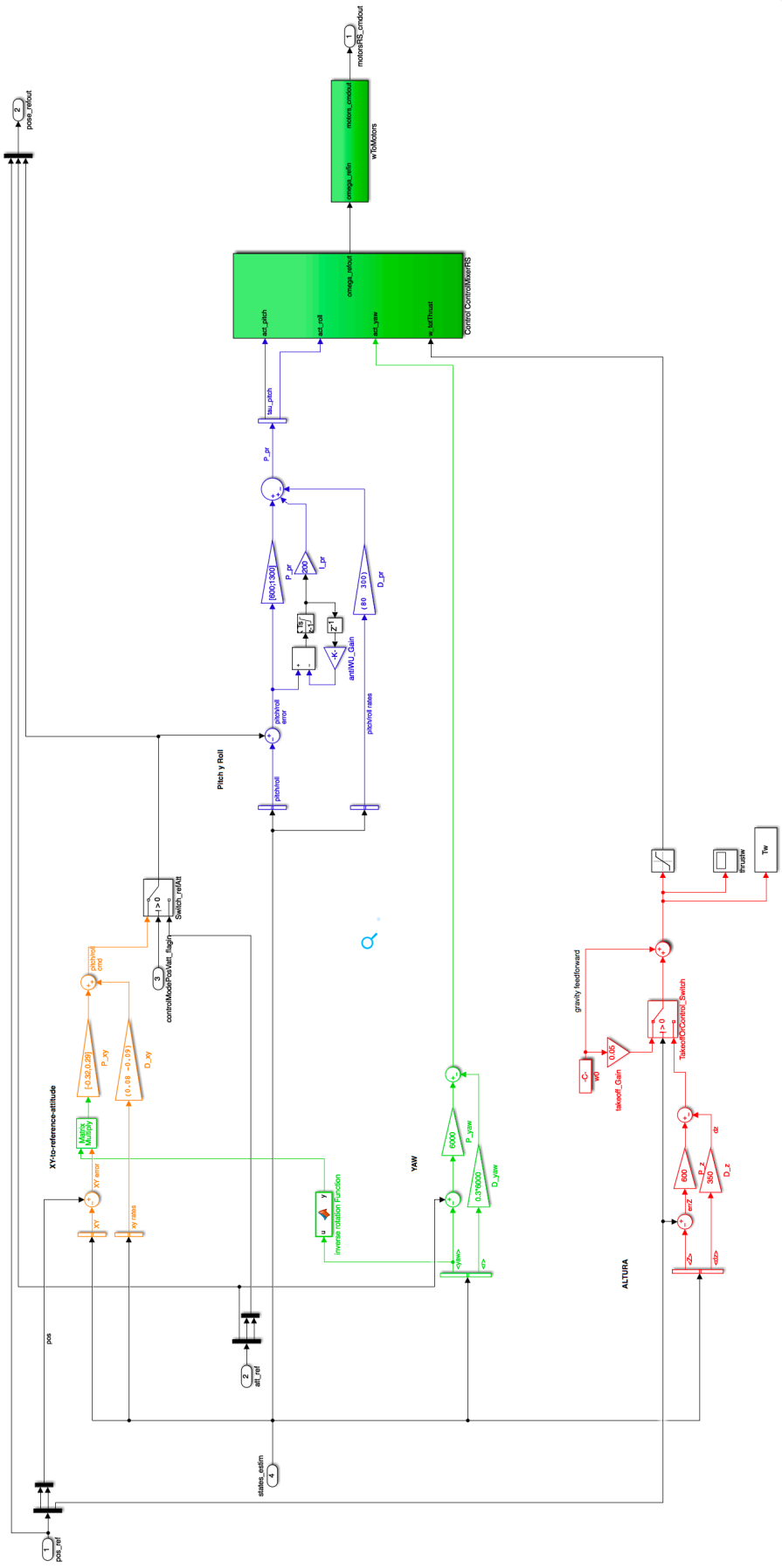



**Figura 4-5 Controlador PID utilizado**

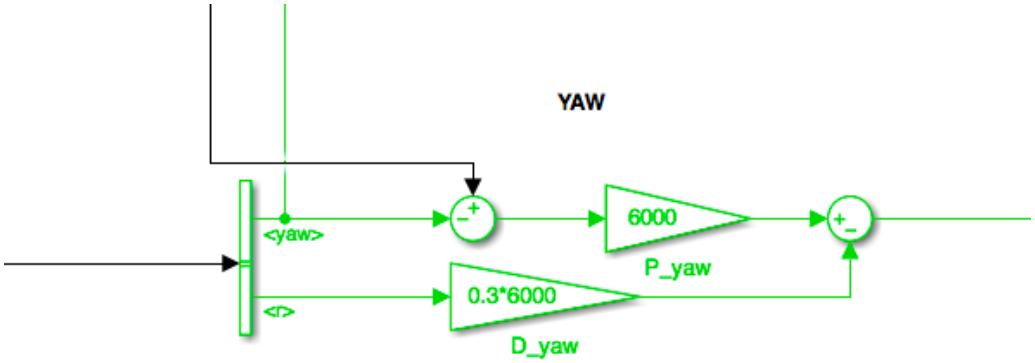



**Figura 4-6 Controlador PD para Yaw**

- Altura: En este controlador se distinguen dos partes, una que se usará únicamente durante el despegue del vehículo y otra que se utiliza para el control de altura en estado normal. Para el despegue se aplica un empuje constante durante un tiempo predeterminado, unos dos segundos. Durante el control de altura en el resto del vuelo se aplica un controlador PD de forma similar a los descritos, un término proporcional con el error entre la posición actual y la referencia y un término derivativo con la velocidad en el eje z.

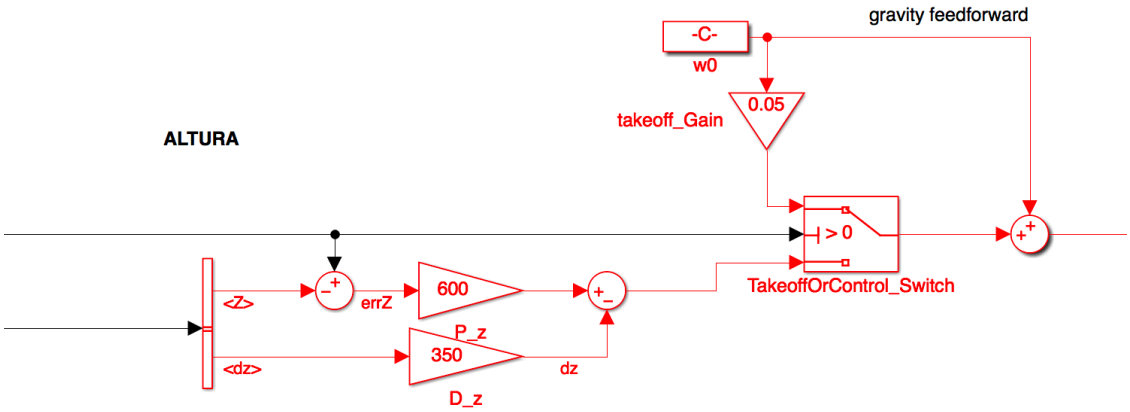



**Figura 4-7 Controlador PD para altura**

- Pitch y roll: En este último controlador, a diferencia de los anteriores, se aplica un PID. Se calcula el error entre el comando de pitch y roll mencionado en el control de posición XY y las componentes del vector de estado correspondientes. A este error obtenido se le aplica tanto el término proporcional como el integral, por otro lado, las velocidades angulares se multiplican con el derivativo. Destacar que en la parte integral se aplica un sistema de anti wind-up.

  El sistema de anti wind-up entra en juego cuando se produce una saturación de la entrada, cuando esto ocurre la integral del error continúa creciendo. Cuando el error disminuye, el alto valor de la integral impide seguir con un funcionamiento normal, la respuesta se verá retrasada. La solución es que al alcanzar dicha saturación se detiene la integración del error, deteniendo así el aumento del término integral del controlador.

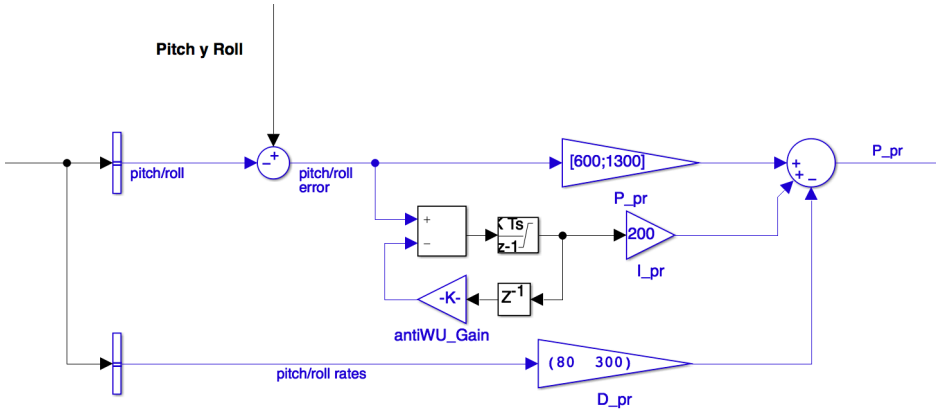



**Figura 4-8 Controlador PID con anti wind-up para pitch y roll**

Las salidas de los controladores de yaw, pitch, roll y altura se envian a un bloque llamado *ControlMixer*, este combina todas las señales aplicándoles límites de saturación obteniendo así, una salida en términos de velocidades angulares. Aún así, esta matriz no se puede enviar directamente a los motores ya que estos necesitan una señal PWM, por lo que se realiza una última transformación.

Explicado el funcionamiento del controlador se procede a realizar una serie de experimentos para comprobar los resultados de este sistema de control en situaciones reales.

### 4.1.1 Control de altura

En este primer experimento se quiere ver como reacciona el controlador ante cambios en la referencia de la altura. Se mostrarán varios resultados con modificaciones en los parámetros para observar los efectos que se mencionaron al comienzo de este punto. Por último, se mostrará una comparación entre la simulación y el vuelo real.

El experimento consiste en enviar una orden de subir hasta 1m en el despegue, en el segundo 5 se cambia la referencia a 1.5m y en el segundo 9 a 0.5m.

Los parámetros escogidos como óptimos para el funcionamiento deseado son:

$$P = 600$$

$$D = 350$$

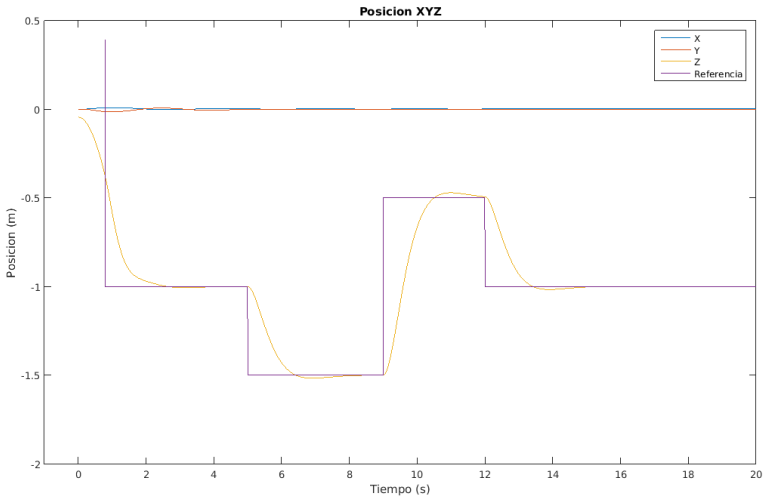



**Figura 4-9 Control de altura PID (1)**

En la Figura 4-9 se representa el controlador con los parámetros mencionados. Se obtiene un tiempo de subida de 2.6 segundos y una sobreoscilación nula, por ello se llega a que estamos ante un sistema sobreamortiguado.

Se podría obtener una respuesta más rápida modificando el término proporcional, esta modificación implica que el controlador será más agresivo a la hora de corregir el error que se está cometiendo. Para la Figura 4-10 se han utilizando los siguientes parámetros:

$$P = 1500$$

$$D = 350$$

Como se puede observar, se obtine una respuesta más rápida, un tiempo de subida de 1.195 segundos, pero por ello se recibe una sobreoscilación de un 14%. El compromiso que existe entre tiempo y error que se mencionó se puede ver claramente en este ejemplo, al conseguir una mejora en uno de los términos se obtiene un peor funcionamiento en cuanto al error. Dependiendo del sistema que se desea controlar y su objetivo se deben poner unas características de funcionamiento deseadas, en este caso no se quiere que exista sobreoscilación para evitar futuras oscilaciones intentando controlar correctamente el sistema porque, aunque en la gráfica de la simulación no lo parece, pequeñas oscilaciones pueden volver la respuesta inestable cuando se prueba en vuelo real.

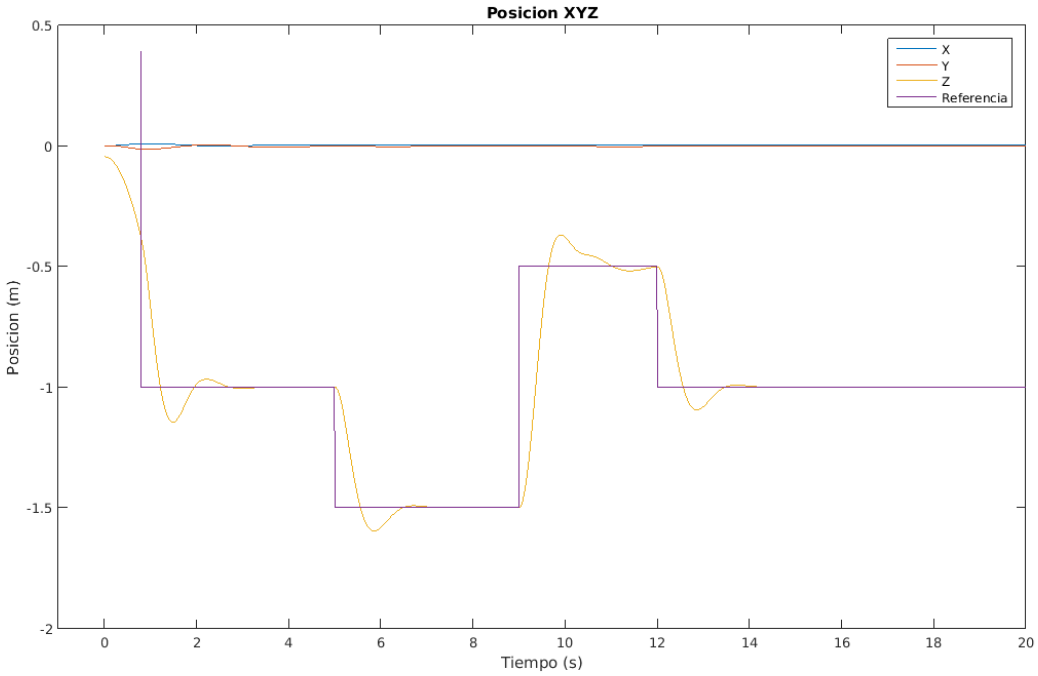



**Figura 4-10 Control de altura PID (2)**

Se podría razonar que aumentando el término derivativo se conseguiría una respuesta algo más lenta, ya que este término se encuentra restando al proporcional, pero más rápida que el controlador escogido mejorando a la vez el problema de la sobreoscilación. Así, en la Figura 4-11 se han utilizado los parámetros:

$$P = 1500$$

$$D = 900$$

Pero el resultado obtenido es ahora más lento que el óptimo escogido, no se tiene sobreoscilación, pero se observa un tiempo de subida de 3.225 segundos.

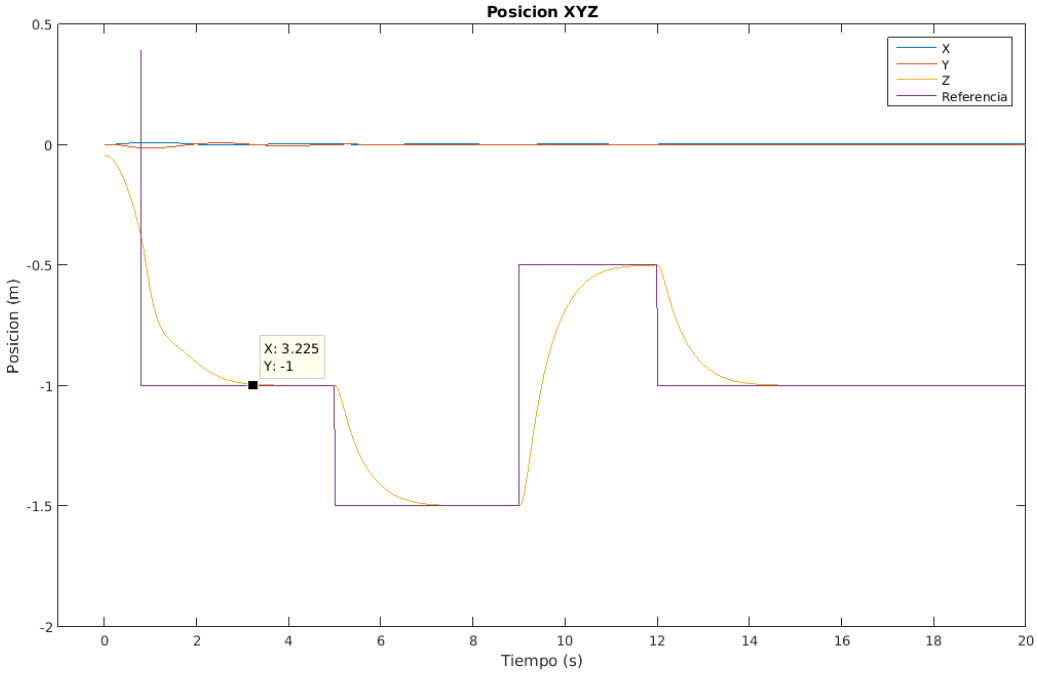



**Figura 4-11 Control de altura PID (3)**

Por último, en la Figura 4-12 se pueden ver los efectos de un mayor término derivativo sin modificar el proporcional. Parámetros usados:

$$P = 600$$

$$D = 900$$

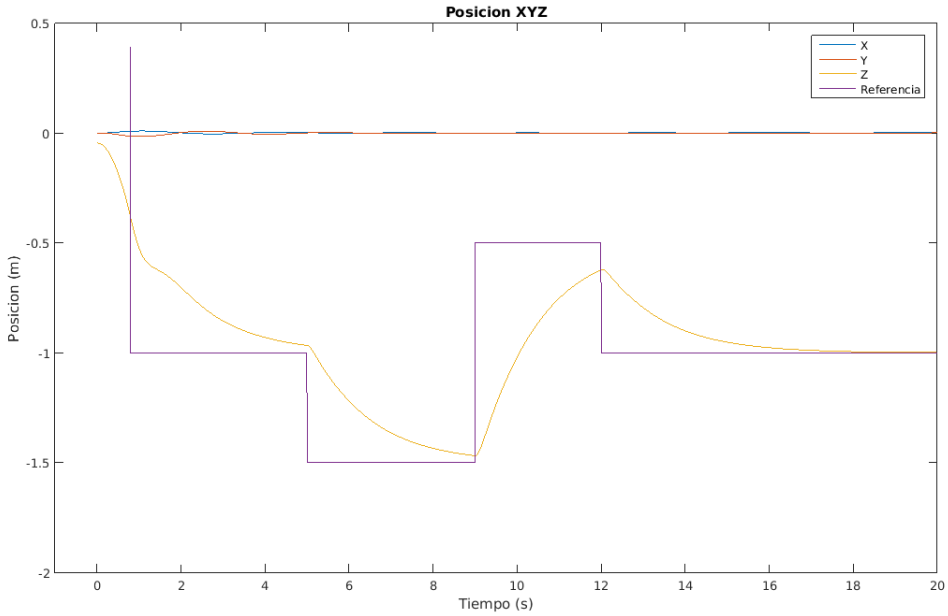



**Figura 4-12 Control de altura PID (4)**

Como se percibe en la representación mostrada, la respuesta del régimen transitorio es mucho más lenta que los controladores anteriores, tanto que no llega siquiera a la referencia deseada.

Si se realiza un experimento similar pero en lugar de simulación se hace en el drone, se obtiene este resultado:

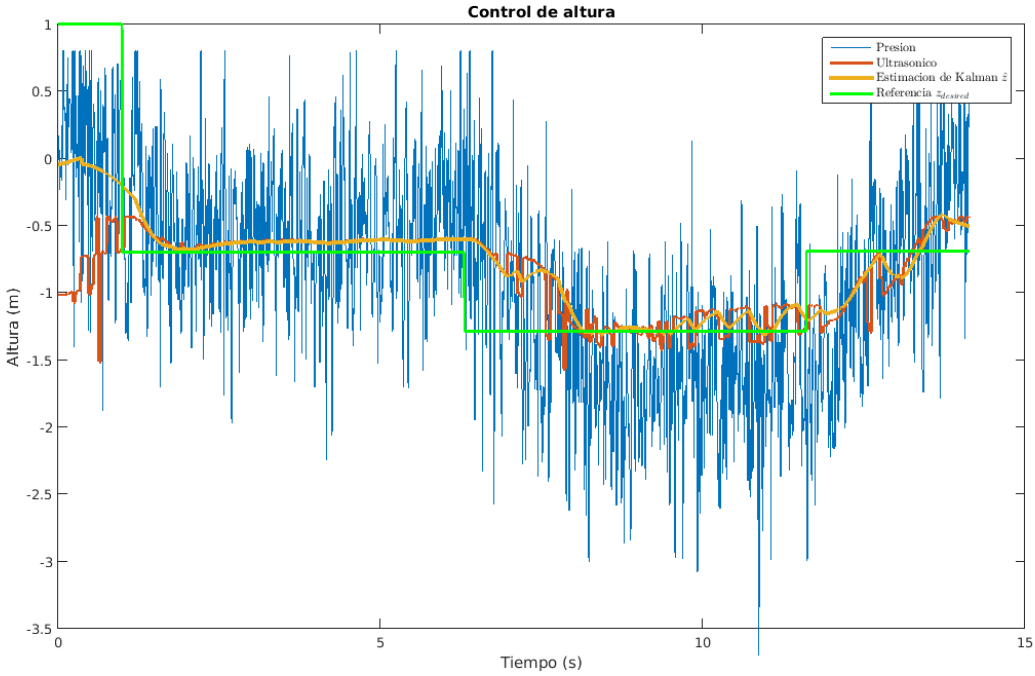



**Figura 4-13 Control de altura en drone con PID**

Se observa claramente como se tiene un resultado distinto al visto en las simulaciones, muchas más oscilaciones y le cuesta mantener el error en régimen permanente a cero. Como ya se ha comentado, en el vuelo real del quadrotor entran en juego algunos factores que el Toolbox de robotica no tiene en cuenta.

### 4.1.2 Control de yaw, pitch y roll

De forma similar al experimento de control de altura, a continuación, se procede a comprobar los resultados del controlador PID para distintos cambios de yaw, pitch y roll. Esto, junto con la prueba anterior, proporcionará una buena base para seleccionar el mejor controlador para futuras aplicaciones que puedan realizarse con este vehículo, como se verán en el apartado 5.

Para este experimento, se darán múltiples referencias en distintos tiempos y se representarán gráficas de los ángulos mencionados. La prueba planteada es la siguiente:

1) Se inicia la simulación con el despegue del drone y se esperan 3 segundos para que se equilibre.
2) Referencia de -0.03 en pitch (equivalente a avanzar) durante 3 segundos.
3) Referencia de 0.03 en roll (movimiento lateral hacía la derecha) durante 3 segundos.
4) Referencia de 0.2 en yaw (giro en sentido horario) durante 3 segundos.

La Figura 4-14 se corresponde con la representación de los ángulos yaw, pitch y roll de este experimento. Como se puede observar, existe un buen comportamiento del controlador ante los cambios de referencia, se anula el error transitorio con velocidad y se mantiene constante durante los 3 segundos que se mantiene y seguidamente, vuelve a la normalidad con facilidad.

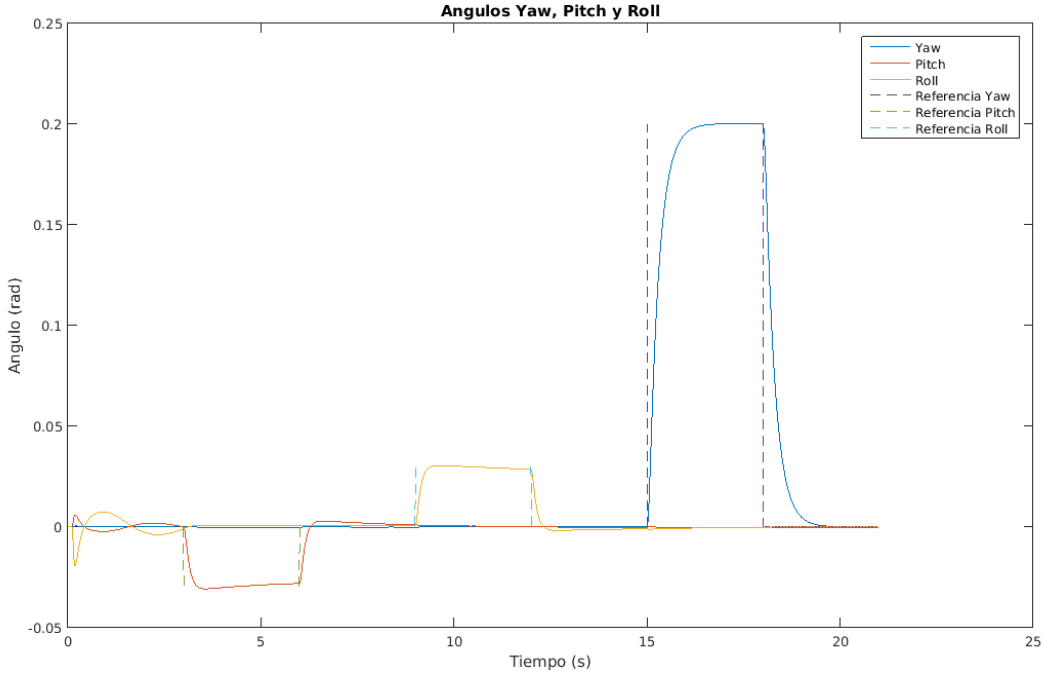



**Figura 4-14 Control de Yaw, Pitch y Roll con PID**

También es buena idea observar como han afectado estos cambios de referencia a la gráfica la de posición del eje Y frente la del eje X (Figura 4-15). Se puede ver que el punto de inicio es el (0,0) y puede parecer que en lugar de avanzar va en diagonal, pero en realidad se desconoce la orientación inicial del vehículo, se puede suponer que se encuentra la línea de la diagonal que sigue, ya que tiene sentido el resultado obtenido a partir de las referencias dadas.

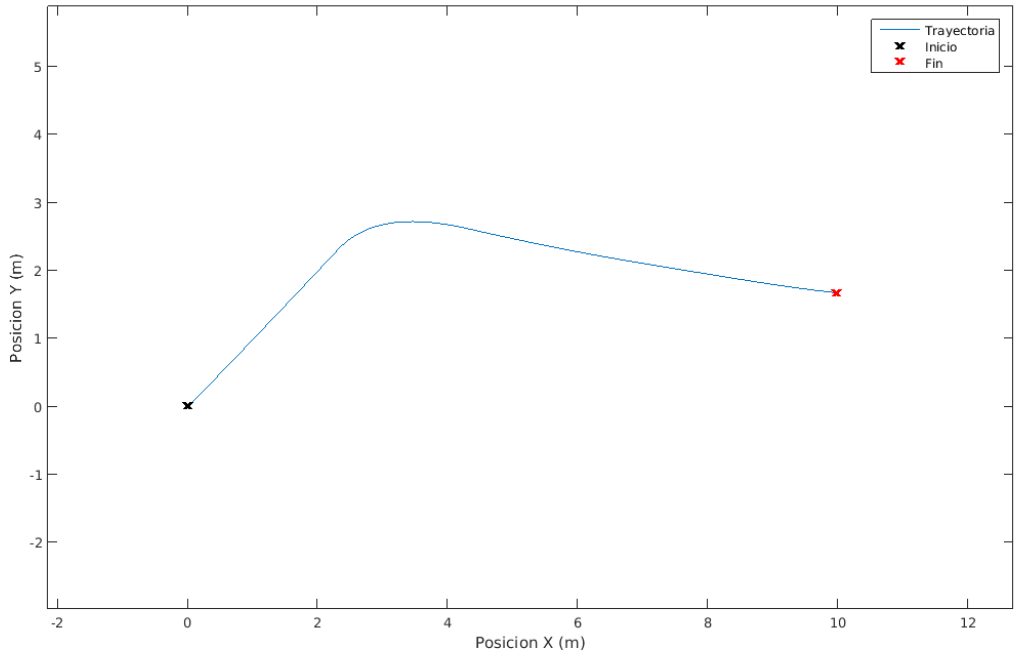



**Figura 4-15 Control de Yaw, Pitch y Roll con PID, Y-X**

Realizando un experimento similar en vuelo real, en lugar de simulación, se obtienen los resultados de la Figura 4-16. Como se puede apreciar, el resultado no se acerca al ideal que se puede ver en la simulación (Figura 4-14). El controlador presenta en todo momento cierta oscilación en cada ángulo de Euler. En esta Figura se da una referencia de Pitch e intenta adaptarse a ella, oscilando a su alrededor.

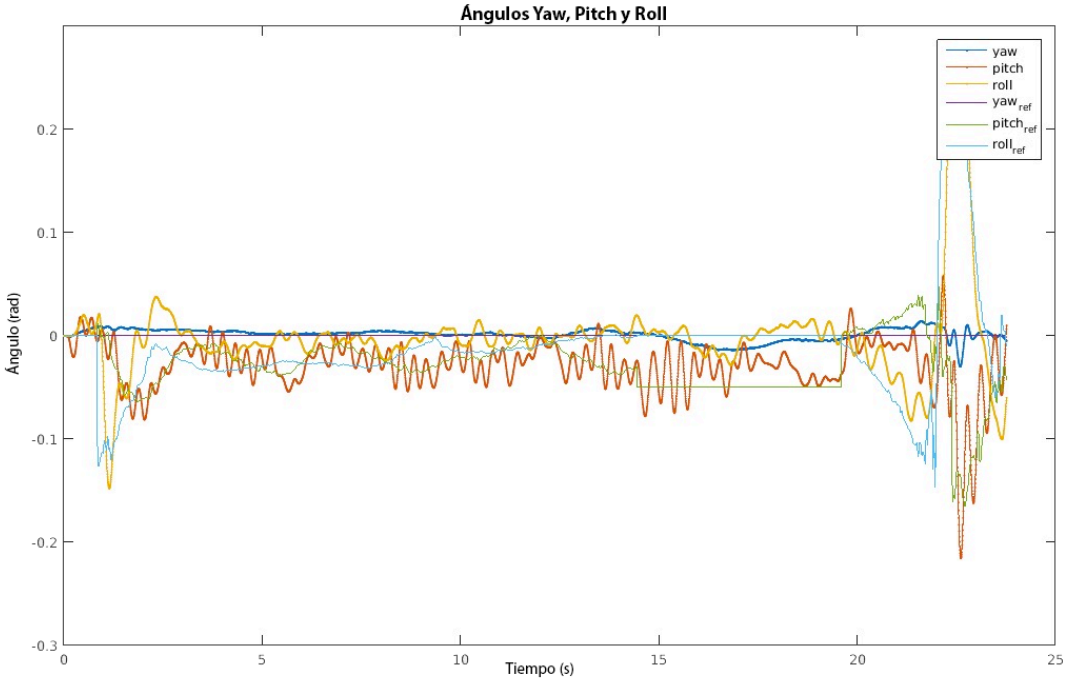



**Figura 4-16 Control de Yaw, Pitch y Roll en drone con PID**

Los cambios bruscos de los ángulos al final de la representación ocurren cuando el vehículo choca con un objeto.

Si se hace zoom a la zona en la que se ordena la referencia de pitch se puede observar mejor como reacciona ante esta (Figura 4-17).

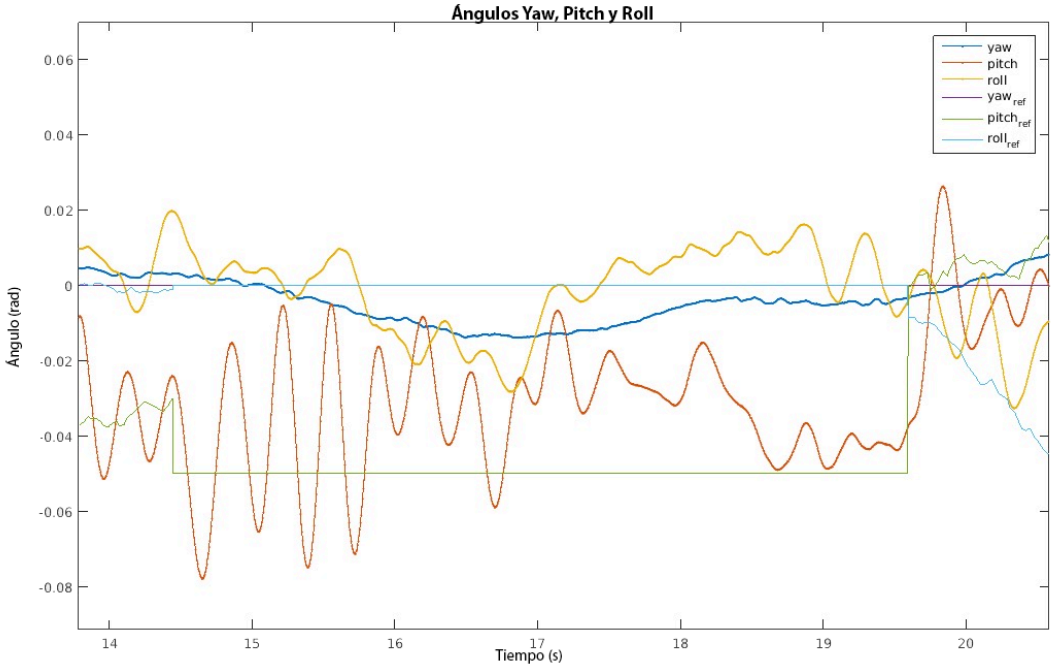



**Figura 4-17 Control de Yaw, Pitch y Roll en drone con PID - Zoom**

Este experimento se corresponde con el video “Control PID” en la carpeta “Videos” del repositorio [2].

### 4.1.3 Seguimiento de trayectoria marcada

Visto ya el control de altura y de los ángulos de orientación, es interesante realizar un último experimento observando la respuesta de la posición de X e Y ante una serie de referencias que simulen una trayectoria fija, para este ejemplo un cuadrado. La figura se ha definido a partir de una serie de señales escalón que generan diferentes movimientos ordenados para pitch y roll. Señalar que cada escalón tiene una duración de tres segundos y que al acabar cualquier referencia se dan dos segundos para que el drone se estabilice. En concreto, los pasos seguidos son los siguientes:

1. Roll con ganancia -0.03, movimiento lateral hacía la izquierda, desde el segundo 5 al 8.
2. Pitch con ganancia 0.03, movimiento hacía atrás, desde el segundo 10 al 13.
3. Roll con ganancia 0.03, movimiento lateral hacía la derecha, desde el segundo 15 al 18.
4. Pitch con ganancia -0.03, movimiento hacía la delante, desde el segundo 20 al 23.

El resultado obtenido se ve en la Figura 4-18. Se puede observar como existen una serie de oscilaciones en el momento que el drone se detiene e intenta equilibrarse en el lugar alcanzado, por eso hay un pequeño tramo despúes de que se ordene el paro.

También existe cierto desvío en la segunda esquina, seguramente provocada por las pequeñas oscilaciones provocadas.

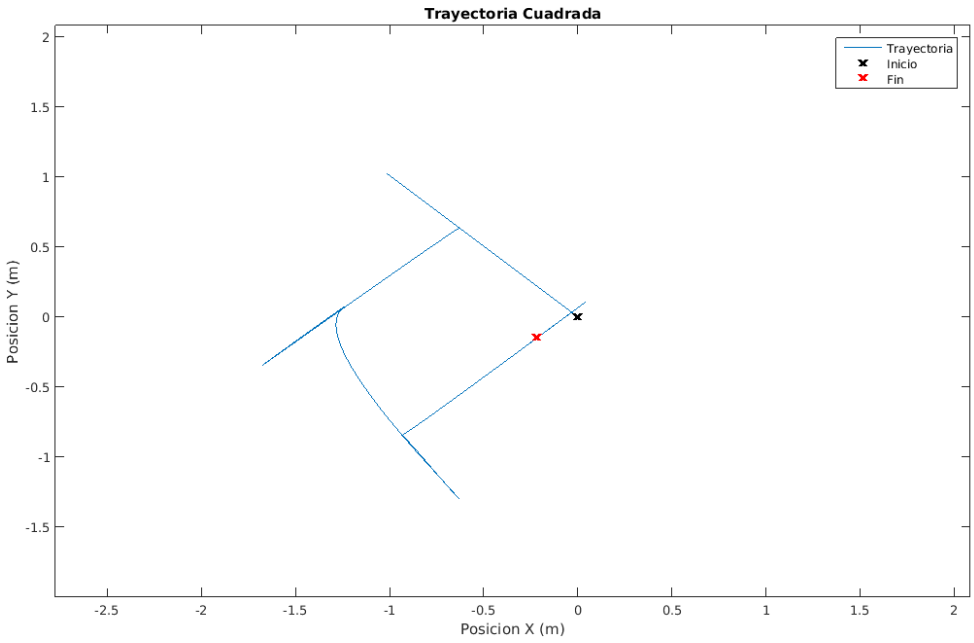



**Figura 4-18 Seguimiento de trayectoria PID**

Si en lugar de mandar el drone a equilibrio entre los cambios de referencias se lleva a puntos de pitch o roll muy bajos, del orden de 0.001, que no llegan afectar al movimiento y permanece quieto, se consiguen mejores giros (Figura 4-19). Esto ocurre porque al tener tanto el roll como el pitch nulos se activa la bandera del control de posición frente al control de orientación y ese cambio puede generar mayores oscilaciones.

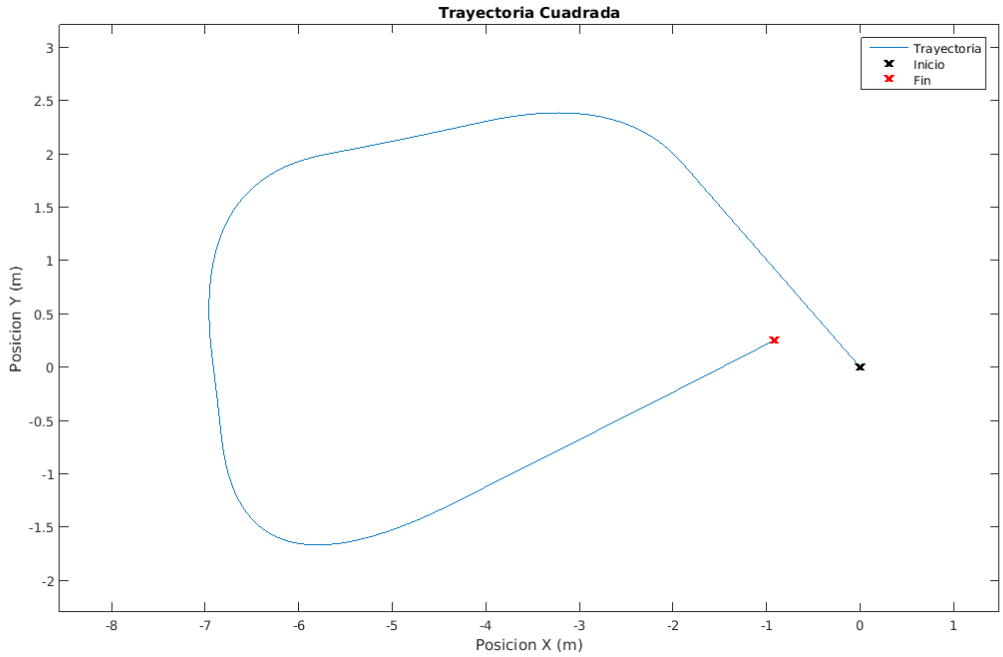



**Figura 4-19 Seguimiento de trayectoria PID (2)**

Aunque se consigue una mayor estabilidad en las curvas, se observa que se ha recorrido una mayor distancia en el eje X y que el punto de fin está más alejado que en la primer prueba, con una distancia en módulo de 0.95 metros. Se puede acordar que esta modificación empeora el funcionamiento del controlador, por lo que sería conveniente mantener la primera.

## 4.2 Controlador LQR

De forma similar al controlador anterior, primero se realizará una explicación teórica de este sistema de control y, seguidamente, se plasmarán los mismos experimentos para poder visualizar la diferencia práctica entre los controladores. El controlador LQR, *Linear Quadratic Regulator*, se basa en la optimización de una función cuadrática. Frente al controlador PID, este sistema de control posee ciertas ventajas:

- Obtención de ley de control en bucle cerrado.
- Coste computacional bajo.
- Control robusto

Las bases teóricas de este controlador se han obtenido de [26] y [24].

Partiendo de las ecuaciones de estado

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B_u}\boldsymbol{u}$$

$$\boldsymbol{z} = \boldsymbol{C_z}\boldsymbol{x}$$

y de la función coste

$$\boldsymbol{J_{LQR}} = \frac{\boldsymbol{1}}{\boldsymbol{2}}\int_{\boldsymbol{0}}^{\boldsymbol{t_f}} [\boldsymbol{x^T Q x} + \boldsymbol{u^T R u}]\boldsymbol{dt} + \frac{\boldsymbol{1}}{\boldsymbol{2}}\boldsymbol{x^T(t_f)P(t_f)x(t_f)}$$

donde $Q \geq 0$ y $R \geq 0$ son matrices simétricas y semidefinidas positivas. Se define $R_{xx} = {C_z}^T Q C_z$.

Se desea buscar la entrada $u\ \forall t\ \in [t_o, t_f]$ de forma que se minimice la función de coste. Por lo que este problema de control consiste más bien en un problema de optimización.

La forma habitual de manejar un problema de optimización con restricciones es añadirlas al coste con un multiplicador de Lagrange.

$$\boldsymbol{L \triangleq f(x,y) + \lambda c(x,y)}$$

Donde $\lambda$ es el multiplicador de Lagrange. En el caso de que se cumplan las restricciones se tendrá que $L \equiv f$.

Como se ha mencionado, el objetivo sería el de encontrar el mínimo de la función anterior:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} = \frac{\partial L}{\partial \lambda} = 0$$

Teniendo así tres ecuaciones con tres incognitas, al resolver se obtenienen un par de puntos (x, y) que marcarián el mínimo deseado.

De forma similar se procedería con la optimización de la función coste del LQR, con el inconveniente de que al incluir integración en su interior se complicaría el procedimiento. Siguiendo el mismo camino, se añadirían las restricciones al coste, obteniendo la función Hamiltoniana:

$$\boldsymbol{H = \frac{1}{2}\left(x^T R_{xx} x\ +\ u^T R u\ \right) + p^T (Ax + B_u u)}$$

Donde $p$ equivale al multiplicador de Lagrange.

Las condiciones necesarias para la optimización serían:

1. $\dot{x} = \frac{\partial H^T}{\partial p} = Ax + B_u u$ , con $x(t_o) = x_o$
2. $\dot{p} = \frac{\partial H^T}{\partial x} = -R_{xx} x - A^T p$, con $p(t_f) = P_{tf} x(t_f)$
3. $\frac{\partial H}{\partial u} = 0 \Rightarrow Ru + {B_u}^T p = 0 \Rightarrow u = -R^{-1} {B_u}^T p$

Así, sustituyendo la ecuación 3 en la 1:

$$\boldsymbol{\dot{x} = Ax - B_u R^{-1} {B_u}^T p}$$

y combinándola con la ecuación 2 se obtiene el siguiente sistema:

$$\begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{p}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{-B_u R^{-1} {B_u}^T} \\ \boldsymbol{-{C_z}^T Q C_z} & \boldsymbol{-A^T} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{p} \end{bmatrix}$$

Donde $\begin{bmatrix} \dot{x} \\ \dot{p} \end{bmatrix}$ es la matriz Hamiltoniana que describe la dinámica en bucle cerrado de x y p.

Las variables x y p se encuentran vinculadas como se ha podido ver, $p(t_f) = P_{tf} x(t_f)$. Esta nueva matriz $P$ debe satisfacer la ecuación algebraica de Riccati.

$$\boldsymbol{0 = A^T P + PA + {C_z}^T Q C_z - P B_u R^{-1} {B_u}^T P}$$

Como se puede ver en la ecuación 3:

$$\boldsymbol{u = -R^{-1} {B_u}^T p = R^{-1} {B_u}^T P x = -Kx}$$

Se define de esta forma la ganancia de control $K$, la cual podría despejarse directamente de la función de Riccati.

La variable de entrada se corresponde con los comandos enviados a los motores y equivalen al empuje de cada motor, en términos que entiende el motor.

$$\boldsymbol{u_{LQR} = [T_1\ T_2\ T_3\ T_4]'}$$

Donde $T_i$ es el empuje que se debe de aplicar al motor i.

Recapitulando, se puede definir el modelo linealizado del sistema como:

$$\boldsymbol{\Delta\dot{x} = A\Delta x + B_u\Delta u}$$

y la función coste:

$$\boldsymbol{J = \int_0^\infty (x^T Q x + u^T R u) dt}$$

donde el término $x^T Qx$ es el coste del estado, con peso Q, y $u^T Ru$ es el coste del control, con peso R.

Una vez explicada la teoría en la que se basa este sistema de control, se procede a explicar el código realizado y los bloques de *Simulink* que intervienen. El archivo ***LQRcontrol.m*** se puede encontrar en la carpeta "Archivos Importantes" del repositorio de Github [2].

Como se ha mencionado anteriormente, el problema parte del modelo linealizado del sistema, este modelo puede obtenerse del Toolbox de robótica de Peter Corke, con los parámetros específicos para este drone. Como este modelo en un principio no está linealizado, es necesario utilizar una herramienta de *Matlab* llamada *Linear Analysis,* con la que se obtiene un modelo ya linealizado del que se puede conseguir las matrices A y B. Por otro lado, se puede definir la matriz C como una matriz identidad con el mismo numero de componentes que el vector de estado, es decir, 12 componentes.

Para calcular las matrices de pesos Q y R de la función coste se utiliza la regla de Bryson. Esta indica que las matrices de pesos pueden ser definidas como una relación entre el peso que se le da a un estado y su valor máximo posible de forma normalizada. Así, se definen las matrices diagonales Q y R como:

$$\boldsymbol{Q} = \begin{bmatrix} \dfrac{\boldsymbol{\alpha_1}^2}{(\boldsymbol{x_1})^2_{max}} & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \dfrac{\boldsymbol{\alpha_n}^2}{(\boldsymbol{x_n})^2_{max}} \end{bmatrix} con\ n = 12\ estados$$

$$\boldsymbol{R} = \boldsymbol{\rho}\begin{bmatrix} \dfrac{\boldsymbol{\beta_1}^2}{(\boldsymbol{u_1})^2_{max}} & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \dfrac{\boldsymbol{\beta_n}^2}{(\boldsymbol{u_n})^2_{max}} \end{bmatrix} con\ n = 4\ motores$$

donde el parámetro $\rho$ se utiliza para ajustar el equilibrio entre la ponderación de control y de estado.

Se impone que la suma de cada peso entre la suma total de los pesos debe de ser igual a uno:

$$\sum_i \frac{\boldsymbol{\alpha_i}}{\sum_i \boldsymbol{\alpha_i}} = \boldsymbol{1}$$

**Tabla 10 Valores máximos para la regla de Bryson**

| Estado | Valor máximo |
|---|---|
| **Posición** | 0.5 |
| **Ángulos yaw, pitch y roll** | 0.3 |
| **Velocidades lineales** | 1 |
| **Velocidades angulares** | 1 |
| **Comando máximo para motores** | 500 |

**Tabla 11 Pesos de los estados en LQR óptimo**

| Peso | Valor |
|---|---|
| **Posición X, Y** | 0.0833 |
| **Posición Z** | 4.1666 |
| **Ángulos yaw, pitch y roll** | 0.0583 |
| **Velocidades lineales** | 0.0167 |
| **Velocidades angulares** | 0.1333 |
| **Rho** | 0.01 |

Definiendo estos parámetros se obtienen las siguientes matrices de ponderación:

$$Q = \begin{bmatrix} 0.0336 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0336 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.6807 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.0392 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0392 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.0392 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.0034 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0034 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0034 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0269 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0269 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0269 \end{bmatrix}$$

$$R = \begin{bmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0.2 \end{bmatrix}$$

Con las matrices ya calculadas, únicamente falta obtener la ganancia de control. Para ello se utiliza el comando de *Matlab lqr( ),* que recibe como entrada las matrices A, B, Q y R y se encarga de encontrar la ganancia que minimiza la función de coste. Obteniendo la siguiente ganancia:

$$K = \begin{bmatrix} -20.498 & 20.498 & -144.9428 & -22.1404 & 98.1932 & 96.1088 & -20.1785 & 19.9818 & -87.0949 & 19.8854 & 20.8355 & -33.3856 \\ 20.498 & 20.498 & 144.9428 & -22.1404 & -98.1932 & 96.1088 & 20.1785 & 19.9818 & 87.0949 & 19.8854 & -20.8355 & -33.3856 \\ 20.498 & -20.498 & -144.9428 & -22.1404 & -98.1932 & -96.1088 & 20.1785 & -19.9818 & -87.0949 & -19.8854 & -20.8355 & -33.3856 \\ -20.498 & -20.498 & 144.9428 & -22.1404 & 98.1932 & -96.1088 & -20.1785 & -19.9818 & 87.0949 & -19.8854 & 20.8355 & -33.3856 \end{bmatrix}$$

Al igual que se ha comentado en el controlador PID, en este sistema de control también se podría utilizar una versión discretizada del sistema. *Matlab* proporciona también un comando para calcular la ganancia que minimiza la función coste para sistemas discretos, bastaría con utilizar *dlqr( ).* El problema es que la ganancia obtenida ha dado un funcionamiento muy diferente al esperado por el controlador, por lo que se ha seguido utilizando la solución en continua, tal y como indica el curso del *MIT* que trata todo lo aquí aplicado.

En cuanto al diagrama de bloques de *Simulink* (Figura 4-20) el funcionamiento es bastante sencillo. El bloque del controlador recibe como entrada: referencia de posición, bandera de despegue, referencia de ángulos de orientación, bandera de control de posición frente a control de ángulos y la estimación del vector de estado.

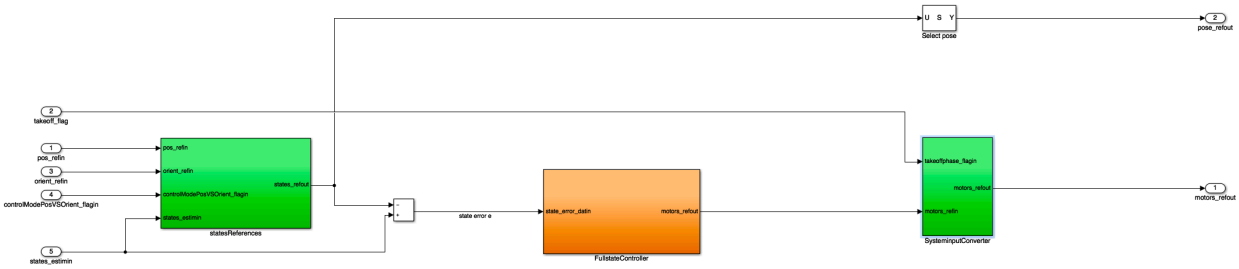



**Figura 4-20 Diagrama de bloques general de LQR**

Un primer bloque, *statesReferences*, selecciona el vector de estado que se utilizará para calcular el error junto con el vector de entrada, dependiendo de si está activado el control de posición o el de orientación. Este vector de errores se pasa al controlador en si (Figura 4-21) que aplica la ley de control $u = -Kx$.

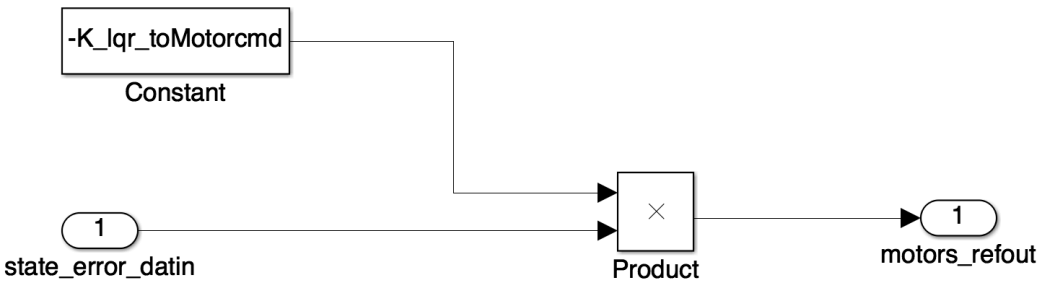



**Figura 4-21 Ley de control para LQR**

A la salida obtenida se le aplica una serie de conversiones para obtener una señal adecuada para enviar a los motores, esto se hace en el bloque *SystemInputConverter*. En ese bloque, también, se genera un empuje constante en el caso que esté activado la bandera de despegue que será sustituido por lo que calcule el controlador en cuanto se desactive dicho valor.

### 4.2.1 Control de altura

En este apartado se procede a realizar el mismo experimento que se ha hecho con el controlador PID para poder comprobar las diferencias entre los sistemas de control. Por ello, las condiciones de la prueba son idénticas, cambios de referencia del mismo tamaño y del mismo tiempo.

Para este experimento, basta con modificar el peso de la posición del eje Z para intentar conseguir distintos efectos del controlador. Para que los parámetros de ponderación estén equilibrados, es común dar el mismo a los estados que se encarguen de la misma tarea, por ejemplo, posición XYZ. De esta forma, se asigna como peso de cada eje $\frac{0.25}{3}$ y se obtiene el siguiente resultado.

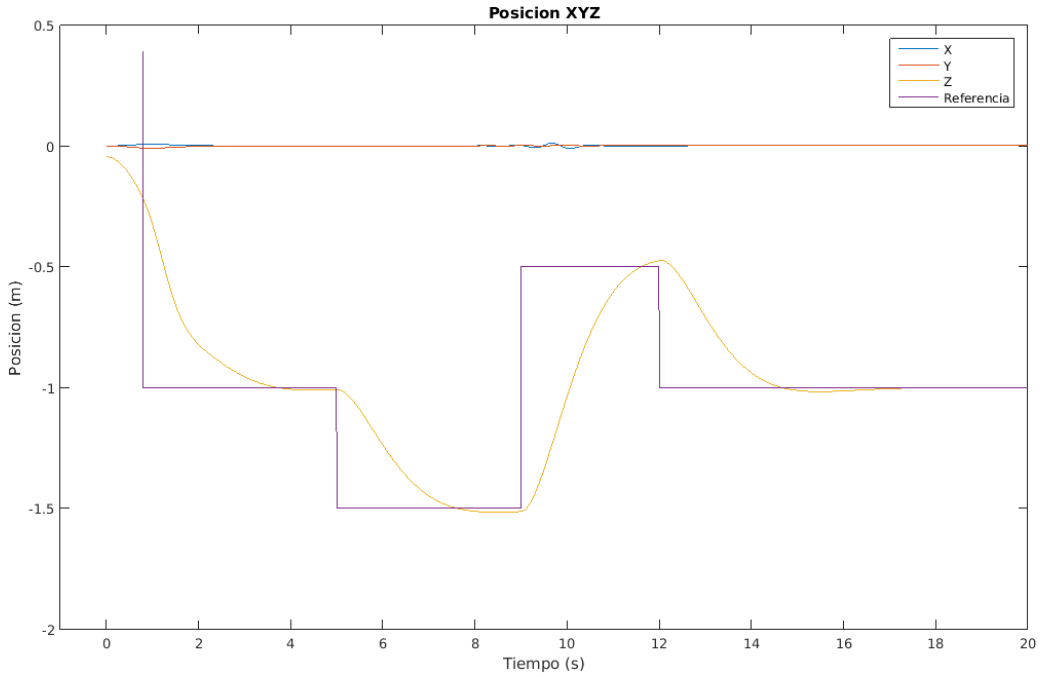



**Figura 4-22 Control de altura LQR (1)**

Obteniendo un tiempo de subida de 4.05 segundos, mucho más lento que los controladores analizados anteriormente. Como se esta intentando controlar los cambios de altura, eje Z, se podría intuir como una buena modificación dar un mayor peso al valor de la ponderación de la posición en el eje z, por ejemplo $\frac{0.25}{3} * 50$.

Se observa una clara mejoría en cuanto a la velocidad de subida, teniendo 2.85 segundos de tiempo de subida.

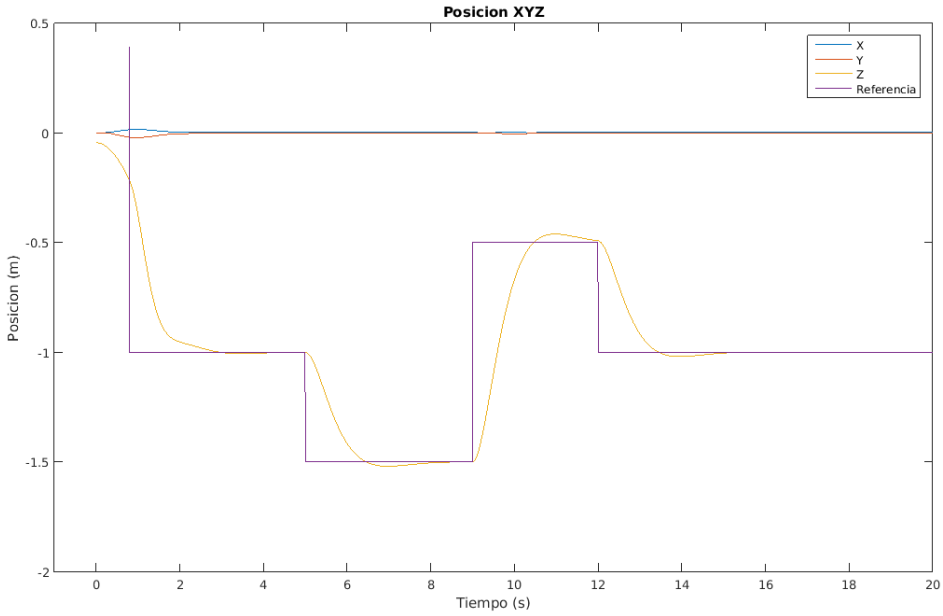



**Figura 4-23 Control de altura LQR (2)**

Si por otro lado, se intentara modificar todos los pesos relacionados con la posición del drone en lugar de alterar únicamente el eje Z, por ejemplo, poniendo los tres pesos a $\frac{0.3}{3}$ . Con esta modificación de un 20% se puede ver como cerca del segundo 9 comienza una pequeña oscilación en el eje X que termina por hacer inestable el sistema, provocando así el choque del vehículo.

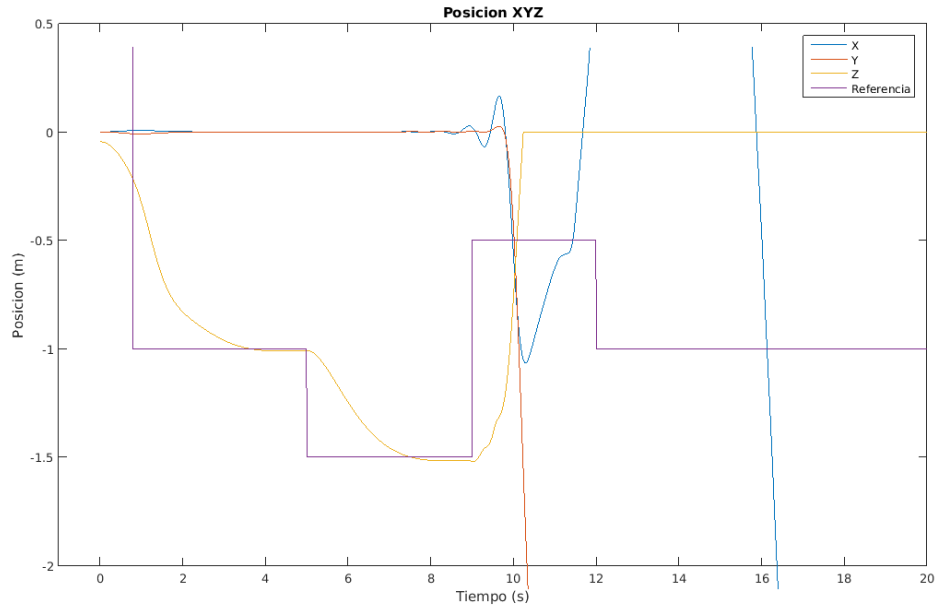



**Figura 4-24 Control de altura LQR (3)**

Para el controlador LQR, el control de altura real del drone es:

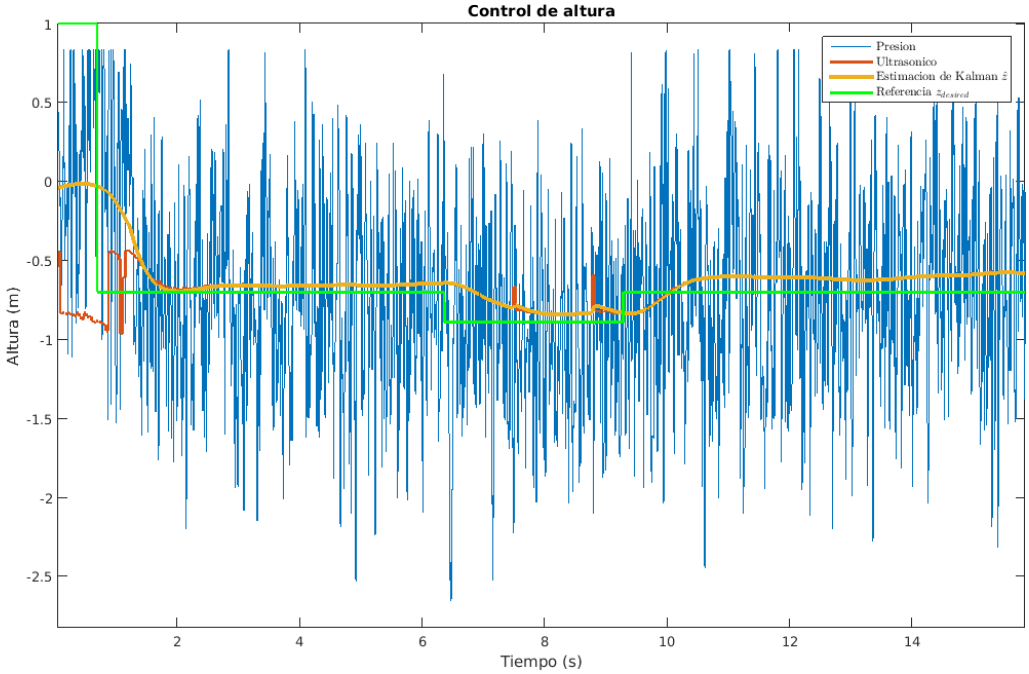



**Figura 4-25 Control de altura en drone LQR**

A diferencia del controlador PID, se pueden observar menos oscilaciones y alguna mejora en el error en régimen permanente, sin ser del todo idóneo.

### 4.2.2 Control de yaw, pitch y roll

Del mismo modo que en el apartado 4.1.2, se realizarán modificaciones en las referencias del yaw, pitch y roll.

Partiendo de los ajustes óptimos del controlador mencionados anteriormente, se obtienen los resultados de la Figura 4-26. Se ve claramente como no es capaz de alcanzar el nivel que se le pide de yaw, esto es debido a que la ponderación asignado al control del ángulo de yaw no tiene el suficiente peso frente a la de los otros dos ángulos.

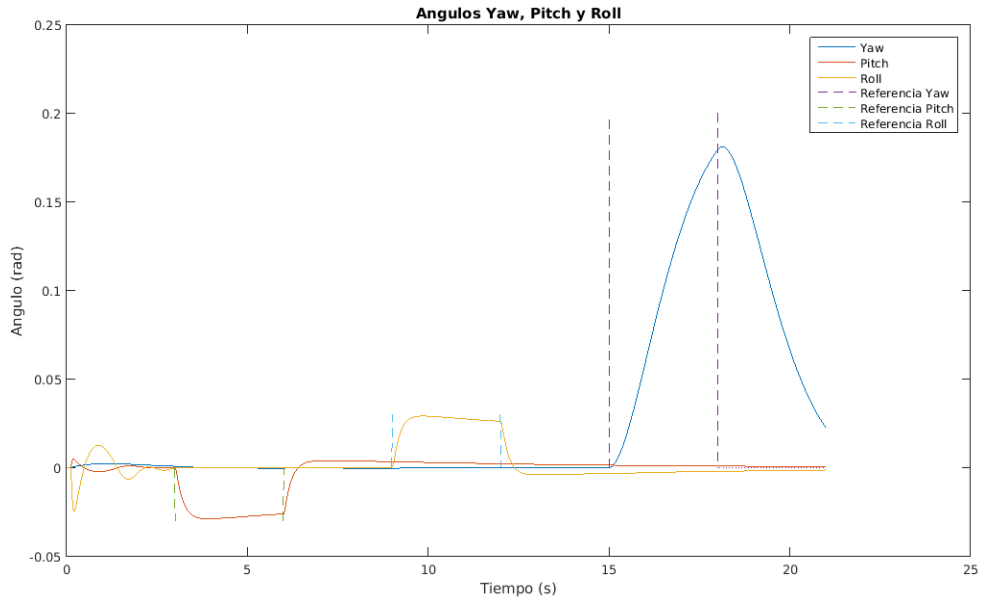



**Figura 4-26 Control de Yaw, Pitch y Roll con LQR (1)**

Si se hace una simple modificación y se multiplica dicho parámetro por tres, quedaría 0.175, se observa una mejoría significante sin alterar el control de los otros ángulos de orientación (Figura 4-27). Aún así, sigue siendo bastante lento comparado con el controlador PID, en el caso de multiplicar el parámetro inicial por un número mayor a tres no se conseguiría mejor resultado, simplemente llegaría cerca del mismo tiempo pero con una sobreoscilación mayor. Por lo que se puede acordar que la modificación realizada es suficiente.

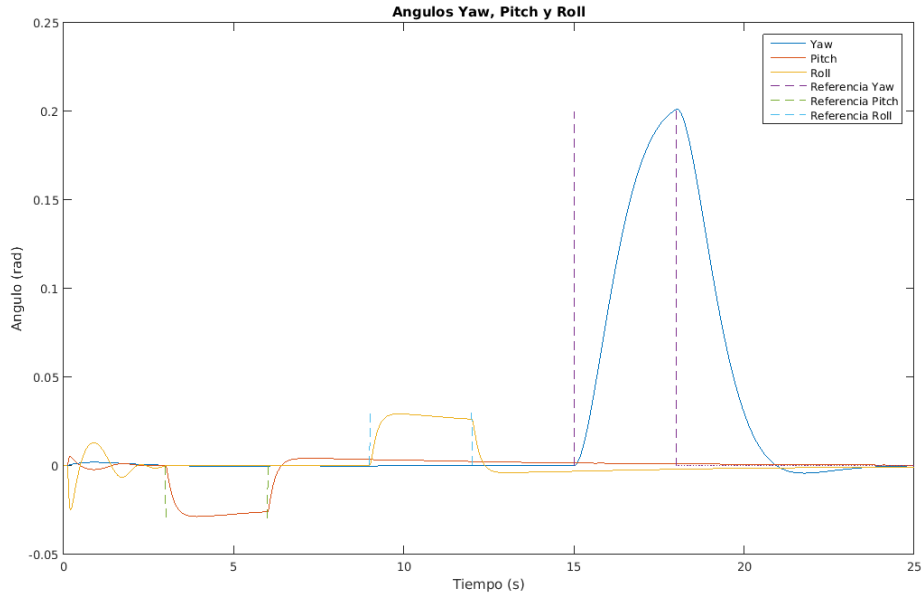



**Figura 4-27 Control de Yaw, Pitch y Roll con LQR (2)**

En la representación de la posición del eje Y frente a la del eje X se observa un comportamiento similar al controlador anterior.

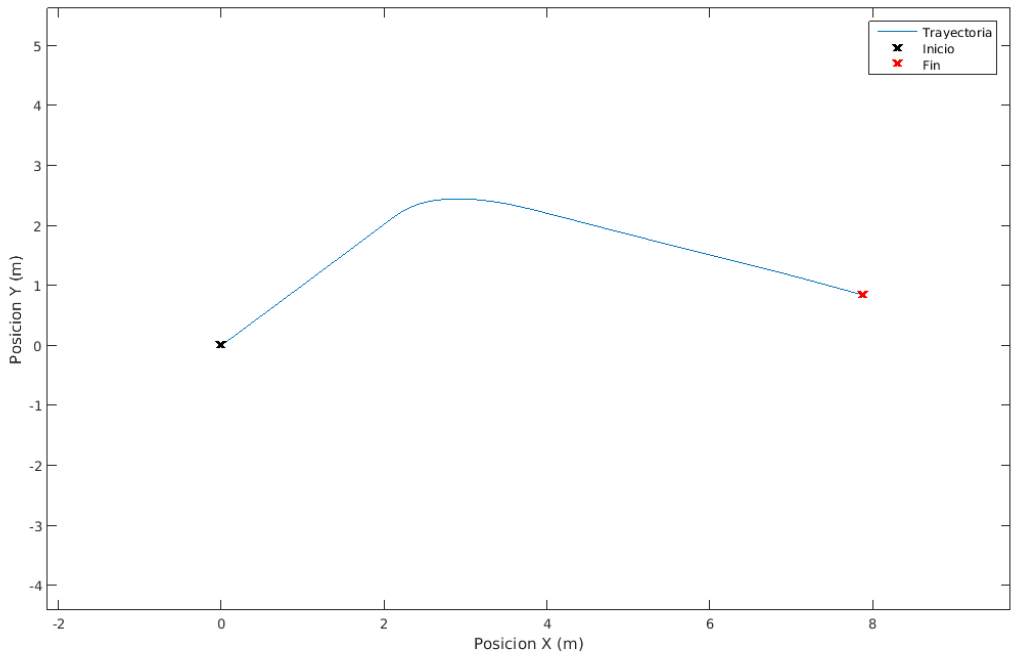



**Figura 4-28 Control de Yaw, Pitch y Roll con PID, Y-X**

Aplicando este control a un vuelo real del drone se observa como el controlador LQR tiene un mayor número de oscilaciones, con una amplitud más grande. Se ordenan varios cambios de referencia de los ángulos de orientación. Para este controlador se le da más peso a las posiciones xyz y a los ángulos de pitch y roll, por

ello se tiene un yaw más descontrolado que no consigue seguir la referencia que se le ordena, en este caso que se mantenga a cero. Se puede coincidir en que el controlador PID tiene una respuesta mejor ante cambios de referencias de ángulos de orientación que el LQR.

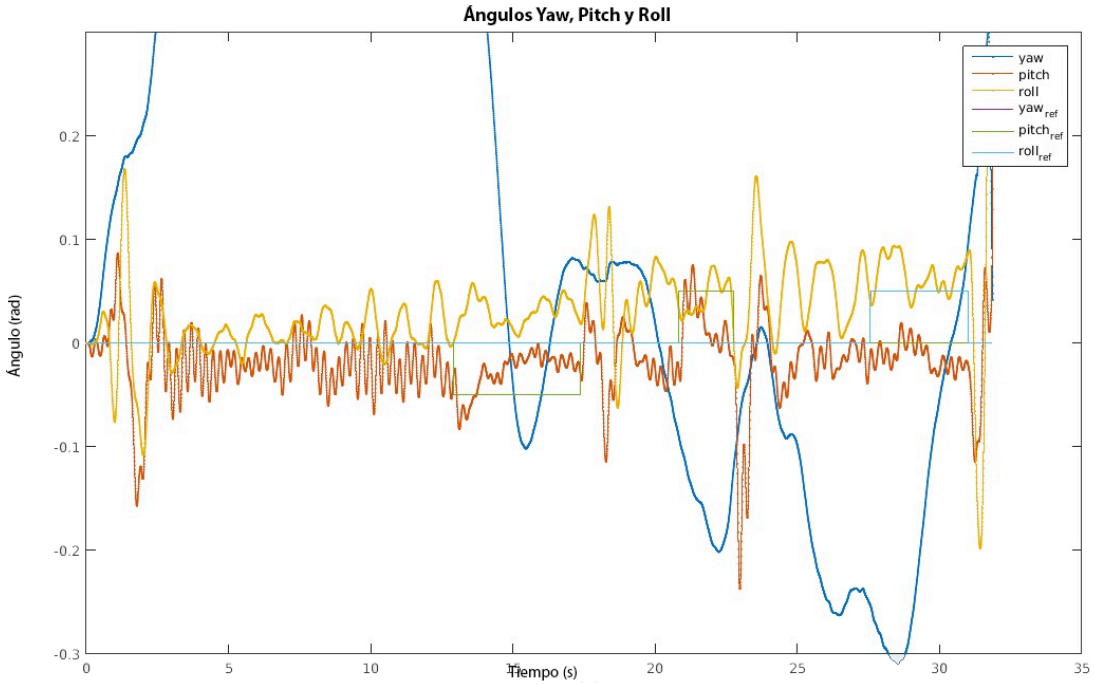



**Figura 4-29 Control de Yaw, Pitch y Roll en drone con LQR**

Este experimento se corresponde con el video "Control LQR" en la carpeta "Videos" del repositorio [2].

### 4.2.3 Seguimiento de trayectoria marcada

Al igual que se hizo con el controlador PID, se mostrará el resultado obtenido tras aplicar a este sistema de control una serie de referencias de pitch y roll que simulan la trayectoria de un cuadrado. En la Figura 4-30 se pueden observar los resultados.

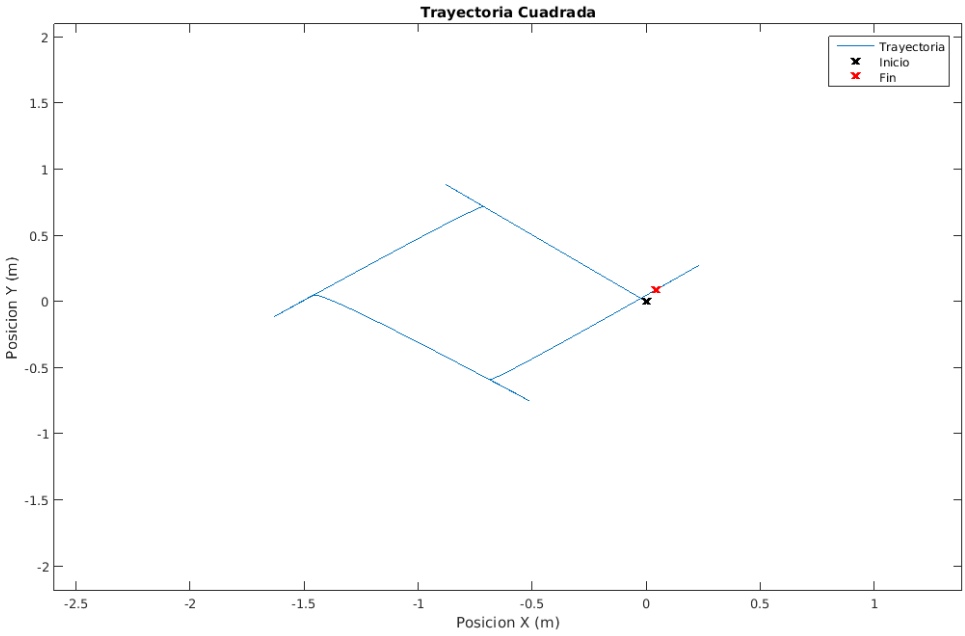



**Figura 4-30 Seguimiento de trayectoria LQR (1)**

Se ve un funcionamiento mejor que el mostrado en el apartado anterior. El punto de inicio y fin se encuentran más cerca, hay menos distancia recorrida tras la orden de paro para equilibrarse y no existe desvío.

Si, al igual que se hizo con el controlador PID, se obliga a que en lugar de detenerse completamente mantenga cierto roll o pitch despreciable:

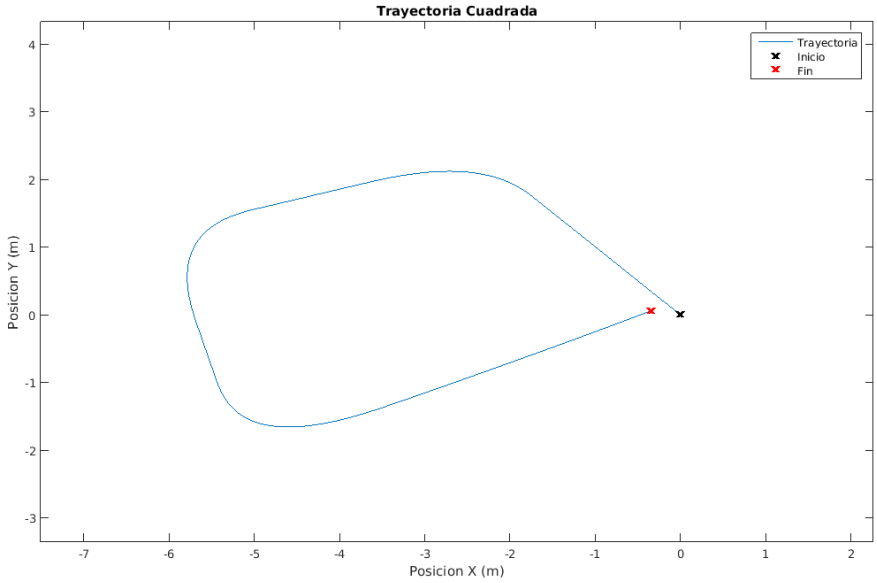



**Figura 4-31 Seguimiento de trayectoria LQR (2)**

De forma similar al PID, se recorre una mayor distancia en el eje X pero se obtiene también una buena aproximación del punto final, con una distancia en módulo de 0.35 metros.

## 4.3 Controlador por ubicación de polos

El último controlador que se analizará es el denominado *Pole Placement*, por asignación de polos. El fundamento de este sistema de control consiste en la elección de una ganancia de control que consiga que la matriz del sistema en bucle cerrado tenga unas propiedades concretas, como puede ser que sea estable y que tenga los polos situados en el lugar deseado del plano complejo. Mencionar que este sistema de control también se puede llamar *Full-State Feedback,* ***FSF****,* para futuras referencias.

Comparando con el controlador PID, se pueden observar varias diferencias:

- Posibilidad de controlar el vector de estado completo, mientras que el PID que se utiliza necesita múltiples estructuras de control para poder corregir todos los errores.
- El controlador por asignación de polos posee un mejor funcionamiento en sistemas no lineales.

Las bases teóricas de este sistema de control se pueden encontrar en [27] y [24].

Se parte de la idea de que la entrada dinámica del sistema se obtiene del sistema:

$$\dot{\boldsymbol{x}}(\boldsymbol{t}) = \boldsymbol{A}\boldsymbol{x}(\boldsymbol{t}) + \boldsymbol{B}\,\boldsymbol{u}(\boldsymbol{t})$$

$$\boldsymbol{y}(\boldsymbol{t}) = \boldsymbol{C}\,\boldsymbol{x}(\boldsymbol{t})$$

donde $u(t)$ es la entrada del sistema en función del tiempo y $x(t)$ es el vector de estado, también dependiente del tiempo. Los polos de este sistema son los autovalores de la matriz A y se pueden obtener como:

$$\boldsymbol{\det(\lambda I - A) = 0}$$

donde $\lambda$ son los autovalores calculados y I es la matriz identidad.

Se quiere utilizar la entrada $u(t)$ para modificar los autovalores de la matriz A y así, poder cambiar la dinámica del sistema hacía un funcionamiento más adecuado.

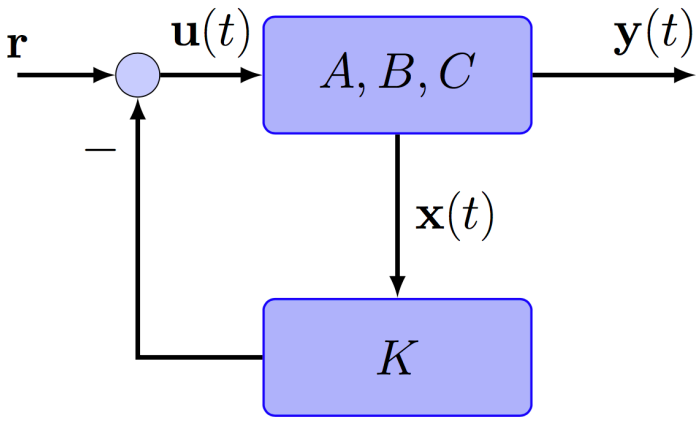



**Figura 4-32 Diagrama del sistema de control FSF**

En la Figura 4-32 se observa una representación del sistema que se está desarrollando. A partir de simples operaciones se puede llegar a una expresión de la entrada del sistema en bucle cerrado:

$$\boldsymbol{u(t) = r - Kx(t)}$$

donde r es la referencia del sistema a controlar y K la ganancia de control.

Desarrollando la dinámica del sistema en bucle cerrado, se obtiene:

$$\boldsymbol{\dot{x}(t) = Ax(t) + B\left(= r - Kx(t)\right) = (A - BK)x(t) + Br = A_{bc}x(t) + Br}$$

$$\boldsymbol{y(t) = Cx(t)}$$

tomando $A_{bc} = (A - BK)$ como la matriz del sistema en bucle cerrado.

Entonces, el objetivo que se quiere conseguir es elegir la ganancia K de forma que $A_{bc}$ tenga las características deseadas, que en este caso serán:

- Sistema estable.
- Polos del sistema en el lugar óptimo.

El sistema del drone a controlar contiene un total de 12 estados y se controlará a partir de 4 motores, por lo que la ganancia de control tiene unas dimensiones de 4x12.

Poniendo un ejemplo para entender mejor el problema presentado, se analizará el siguiente sistema:

$$\dot{x}(t) = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u(t)$$

Se define la entrada como $u(t) = -[k_1 \quad k_2]x(t)$. Entonces se puede calcular la matriz en bucle cerrado como:

$$A_{bc} = (A - BK) = \begin{bmatrix} 1 - k_1 & 1 - k_2 \\ 1 & 2 \end{bmatrix}$$

Eligiendo correctamente $k_1$ y $k_2$ se pueden colocar los autovalores de $A_{bc}$ donde se desee.

A partir de una simple comparación entre los autovalores del sistema en bucle abierto y bucle cerrado, se

pueden obtener los valores de $k_1$ $y$ $k_2$ que satisfagan las características deseadas.

$$\det(\lambda I - A_{bc}) = \lambda^2 + (k_1 - 3)\lambda + (1 - 2k_1 + k_2) = 0$$

Si por ejemplo, se desea situar dos polos $\lambda_1 = -5$ y $\lambda_2 = -6$.

$$(\lambda + 5)(\lambda + 6) = \lambda^2 + 11\lambda + 30$$

Igualando los términos de orden correspondientes con los de la ecuación de los autovalores en BC se obtienen los valores de las ganancias que consigen dicho propósito.

$$\begin{cases} k_1 = 14 \\ k_2 = 57 \end{cases}$$

Como se puede razonar, realizar este mismo procedimiento para un sistema con 12 estados puede ser una tarea compleja y que no es rápida de realizar. Por ello, se hace uso de herramientas y funciones proporcionadas por *Matlab*.

A continuación, se explicará el funcionamiento del código encargado de obtener la ganancia de control para este sistema, esto se encuenta en el archivo ***asignacionPolos.m***.

Para comenzar, al igual que en el controlador LQR, es necesario utilizar una linealización del sistema que se quiere controlar, para ello se utiliza los resultados ya obtenidos en el punto anterior. Teniendo ya esta linealización, se pueden obtener las matrices A y B del sistema.

Se quiere encontrar ahora las matrices desvinculadas del sistema, para poder, así, tener los estados sin que sean dependientes del resto. Un método muy usado de desvinculación es la forma canónica de Jordan, se utiliza cuando existen varios estados con los mismos autovalores. Para ello, se consiguen las matrices A y B a partir de la dinámica del sistema y se introduce A en la función *jordan( )* de *Matlab*, esta devuelve la forma de jordan y una matriz de cambio de base V, de forma que se cumple:

$$\boldsymbol{A = VJV^{-1}}$$

Ahora se tiene un nuevo vector de estado desvinculado:

$$\boldsymbol{x_{dec} = {V_{eig}}^{-1} x}$$

donde $V_{eig} = \begin{bmatrix} \alpha_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \alpha_n \end{bmatrix} V$, con $\alpha_i = \frac{1}{\sum_i columna\ de\ V_i}$.

De forma similar, se obtienen las matrices desvinculadas:

$$\boldsymbol{A_{dec} = {V_{eig}}^{-1} A V_{eig}}$$

$$\boldsymbol{B_{dec} = {V_{eig}}^{-1} B}$$

$$A_{dec} = \begin{bmatrix} 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9.81 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9.81 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B_{dec} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 14577.26 & -2.1474e-12 \\ 0 & 0 & 0 & 0 \\ 14.71 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1.0737e-12 & 10869.57 \\ 0 & 0 & 0 & 0 \\ 0 & 7320.64 & 0 & 0 \end{bmatrix}$$

Con la obtención de las matrices A y B desvinculadas se obtienen una serie de bloques con los estados vinculados entre sí y desvinculados del resto. En concreto se tiene:

- Posición eje X. Para ambas matrices se corresponde con las submatrices de filas 1:4 y columnas 1:4.

$$A_{dec_x} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 9.81 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, B_{dec_x} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 14577.26 & -2.1474e-12 \end{bmatrix}$$

- Posición eje Y. Para la matriz A se corresponde con la submatriz de filas 7:10 y columnas 7:10, en el caso de la matriz B pertenece a la submatriz de filas 7:10 y columnas 1:4.

$$A_{dec_y} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 9.81 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, B_{dec_y} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1.0737e-12 & 10869.57 \end{bmatrix}$$

- Posición eje Z. Para la matriz A se corresponde con la submatriz de filas 5:6 y columnas 5:6, en el caso de la matriz B pertenece a la submatriz de filas 5:6 y columnas 1:4.

$$A_{dec_z} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, B_{dec_z} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 14.71 & 0 & 0 & 0 \end{bmatrix}$$

- Yaw. Para la matriz A se corresponde con la submatriz de filas 11:12 y columnas 11:12, en el caso de

la matriz B pertenece a la submatriz de filas 11:12 y columnas 1:4

$$A_{dec_{yaw}} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, B_{dec_{yaw}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 7320.64 & 0 & 0 \end{bmatrix}$$

A continuación, se escogen los polos para cada estado (Tabla 12).

Seguidamente, se procede a la colocación de los polos de la Tabla 10. Para ello, se utiliza el comando de *Matlab place( )*, este recibe como entrada las matrices A y B del estado así como los polos escogidos y devuelve la ganancia que consigue las características deseadas.

**Tabla 12 Ubicación de polos escogida para FSF**

| Estado | Polos escogidos |
|---|---|
| **Posición X** | $-9 \pm 6i\ ;\ -0.18 \pm 1.8i$ |
| **Posición Y** | $-60; -4\ ;\ -0.1 \pm 2i$ |
| **Posición Z** | $-5; -5.1$ |
| **Yaw** | $-3; -3.1$ |

De esta forma, se tiene cuatro ganancias, una por estado a controlar, y se combinan para obtener la ganancia de control.

$$K_{FSF} = \begin{bmatrix} 0 & 0 & 1.734 & 0 & 0 & 0 & 0 & 0 & 0.687 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.00127 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.00083 \\ -0.00268 & 0 & 0 & 0 & 0.00869 & 0 & -0.000706 & 0 & 0 & 0 & 0.00126 & 0 \\ 0 & 0.00903 & 0 & 0 & 0 & 0.0236 & 0 & 0.00286 & 0 & 0.00591 & 0 & 0 \end{bmatrix}$$

En cuanto al diagrama de bloques, es casi idéntico al que se usa en el LQR, con la diferencia de que en lugar de usar la ganancia de dicho controlador se utiliza la calculada por este procedimiento.

### 4.3.1 Control de altura

Igual que se procedió en los puntos anteriores, a continuación, se comenzará con los ensayos para comprobar el funcionamiento del controlador por ubicación de polos.

Para modificar la respues del controlador ante cambios en la referencia de la altura, eje Z, basta con cambiar los polos que se exigen al sistema. Los polos óptimos encontrados para el eje Z, mencionados en la tabla 12, son los dos polos reales $-5; -5.1$. Estos dan el siguiente resultado frente a los cambios de referencia sugeridos en los puntos anteriores (Figura 4-33).

Se puede observar una buena respuesta ante los cambios de referencia, sin sobreoscilación y con un tiempo de subida de 2.55 segundos, mejorando incluso al controlador PID

Realizando un ligero cambio en los polos de la posición del eje Z a $-2; -2.1$ se ve un claro empeoramiento

del control de altura, obteniendo un tiempo de subida de 4.58 segundos (Figura 4-34).

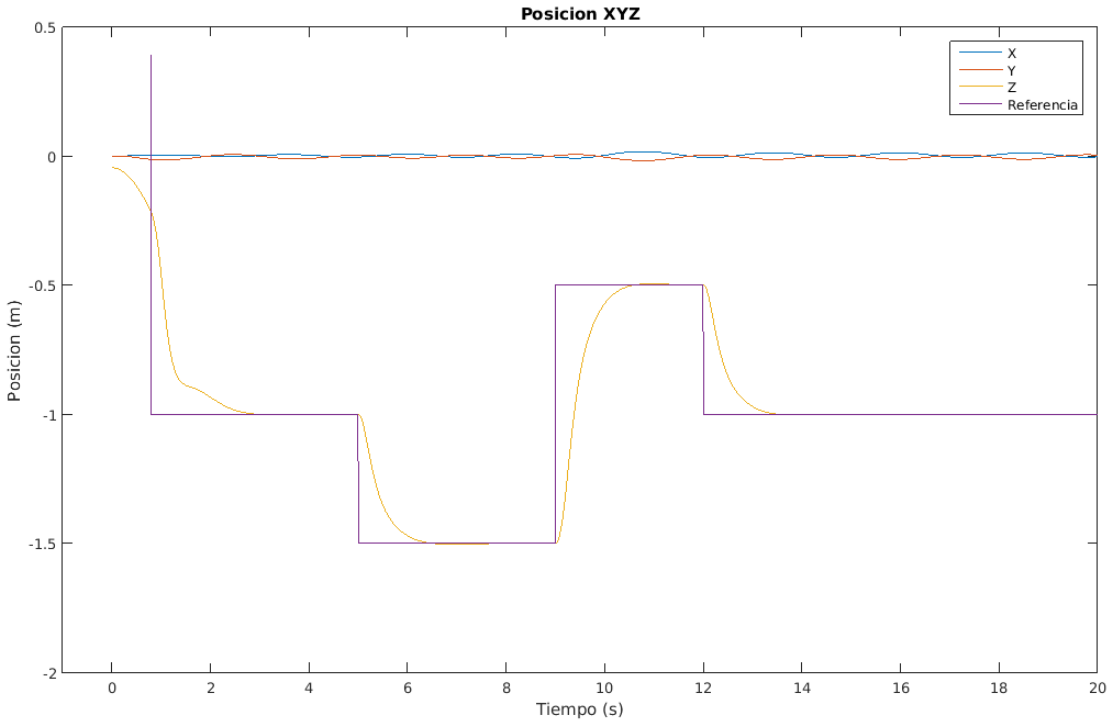



**Figura 4-33 Control de altura FSF (1)**

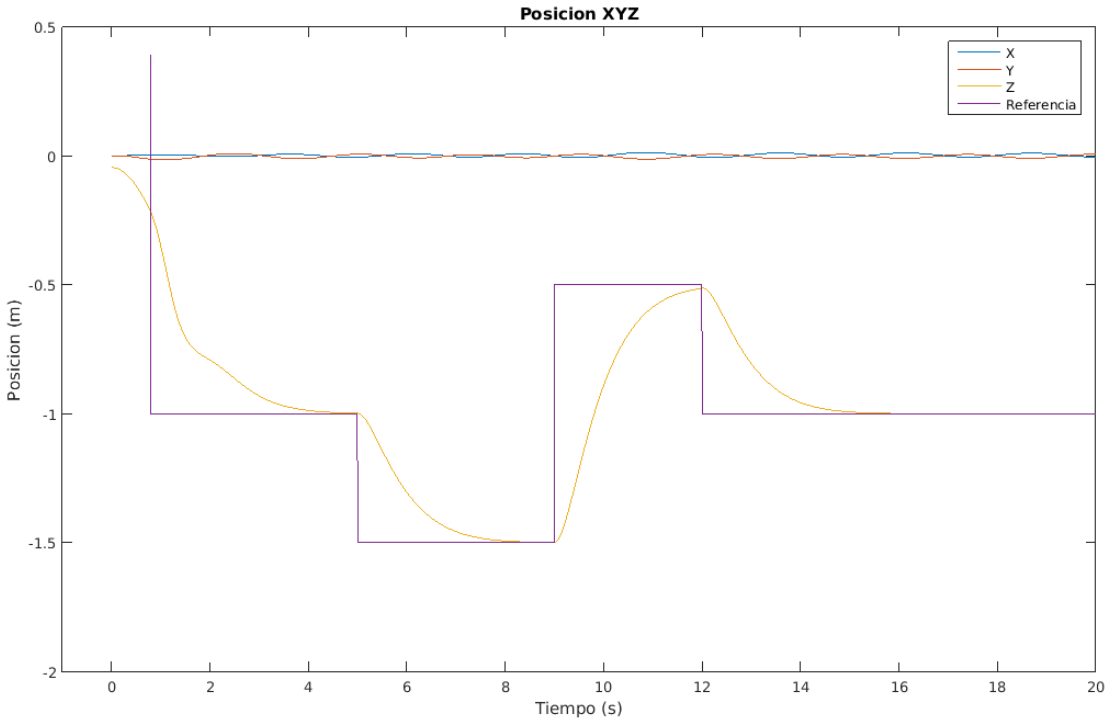



**Figura 4-34 Control de altura FSF (2)**

Si se eligieran unos polos más agresivos, por ejemplo $-12; -12.1$, se consigue un tiempo de subida mucho más veloz, 1.24 segundos, pero esto tiene una consecuencia, un 13.7% de sobreoscilación (Figura 4-35). También es posible ver como se inicia una oscilación en las posiciones X e Y. Por lo que, como se ha mencionado en puntos anteriores, es un compromiso entre velocidad y precisión. Es difícil tener un controlador que sea veloz y preciso. Si además, se utilizasen unos polos demasiado bajos, en este caso, se podría inestabilizar el sistema y generar un choque del vehículo.

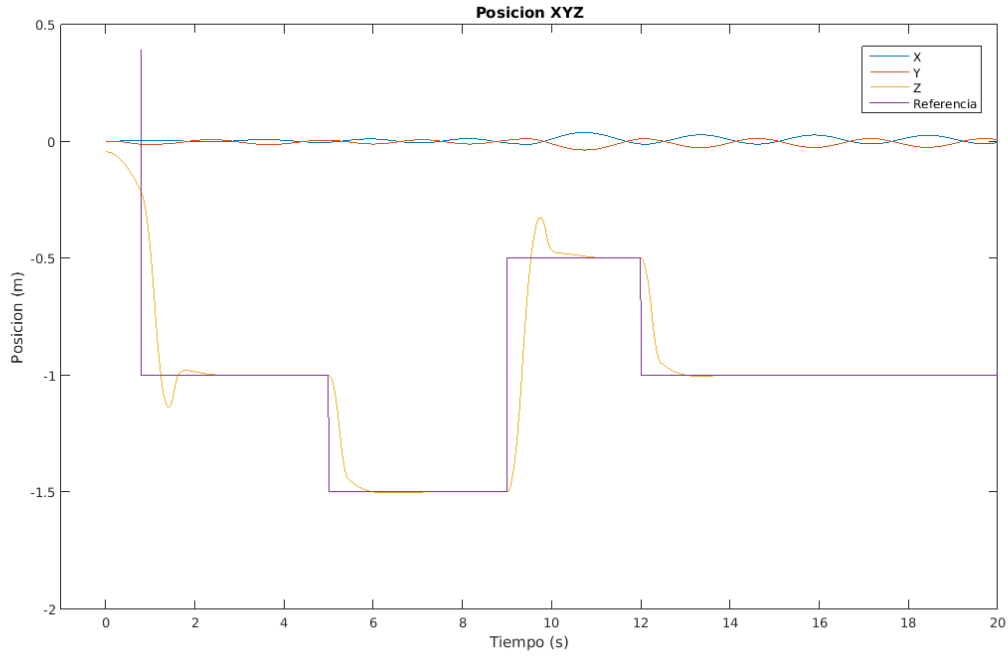



**Figura 4-35 Control de altura FSF (3)**

Para el controlador por asignación de polos se tiene el siguiente resultado para un vuelo real del drone:

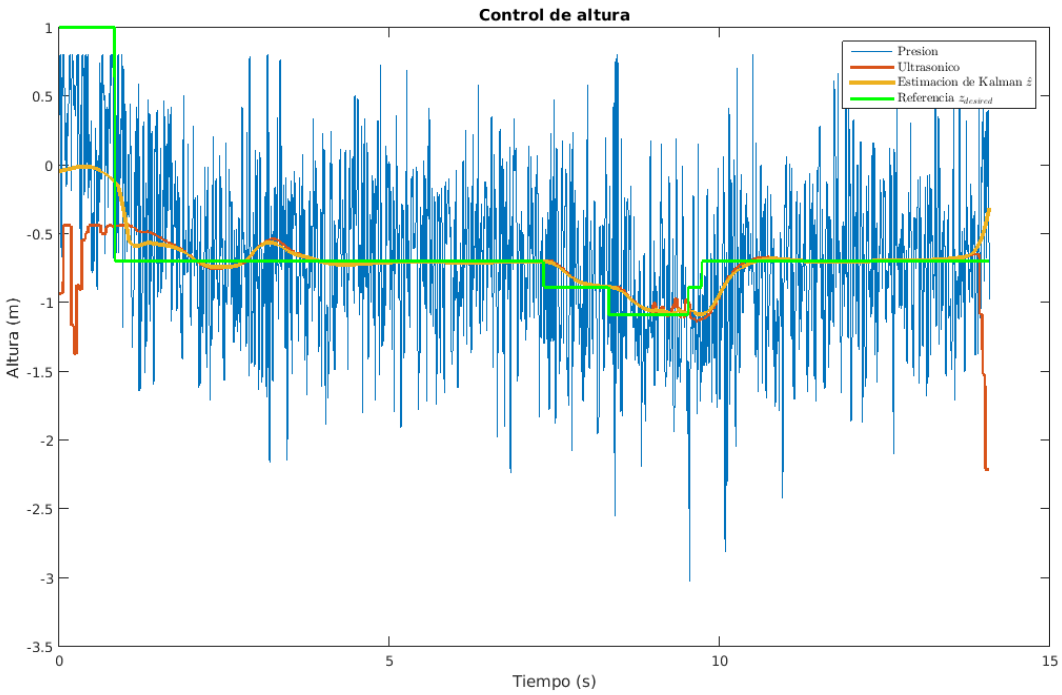



**Figura 4-36 Control de altura en drone FSF**

No hay duda que este controlador tiene un mejor funcionamiento respecto a los demás. Apenas hay oscilaciones y se anula por completo el error en régimen permanente.

### 4.3.2 Control de yaw, pitch y roll

A continuación, se muestran los resultados del experimento del control de yaw, pitch y roll para el controlador por ubicación de polos.

Siguiendo la metodología ya mencionada en puntos anteriores, se obtienen los resultados de la Figura 4-37.

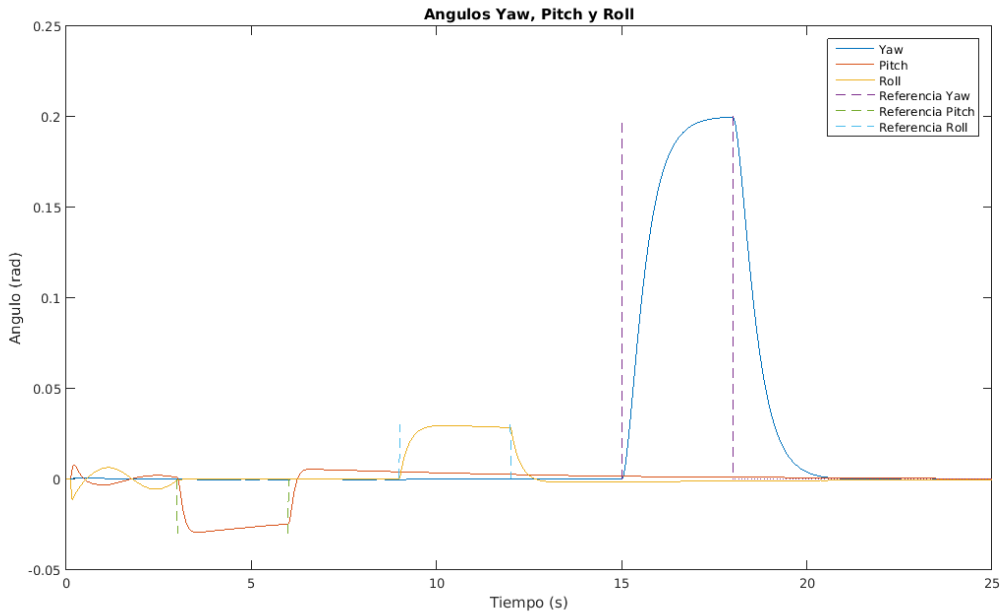



**Figura 4-37 Control de Yaw, Pitch y Roll con FSF (1)**

Aunque la respuesta no es necesariamente errónea, se podría intentar conseguir una respuesta más rápida ante cambios de referencia en yaw. Si se modifica el polo relacionado con el estado del yaw de [-3, -3.1] a [-5, -5.1] (Figura 4-38), se obtiene un funcionamiento más adecuado.

En la Figura 4-39 se muestra una representación de Y frente a X como se ha enseñado en los controladores anteriores. De nuevo, un comportamiento similar a los controladores anteriores.

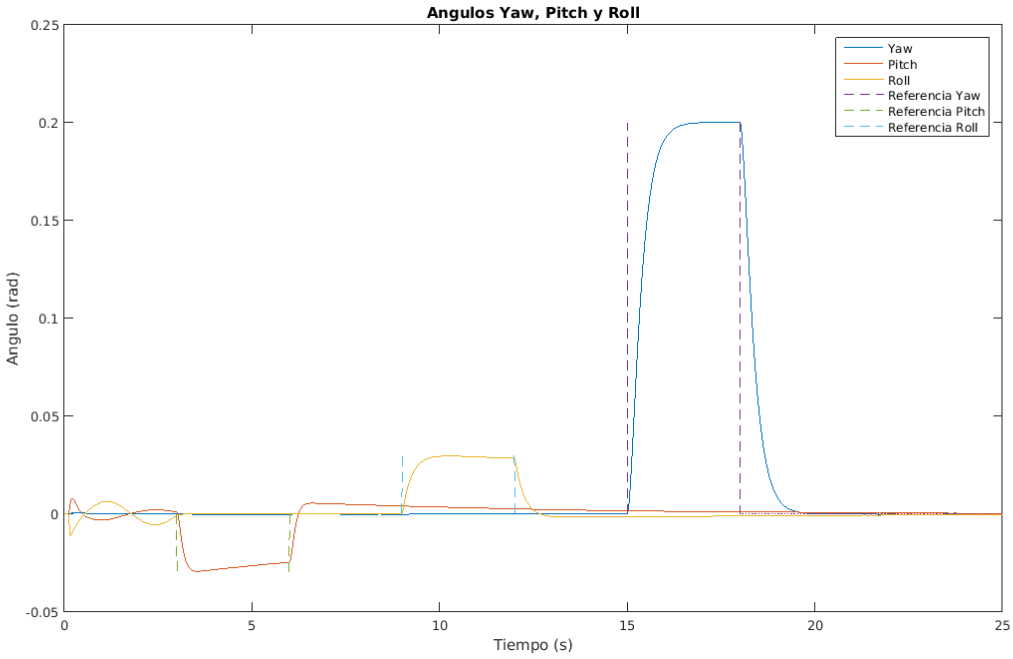



**Figura 4-38 Control de Yaw, Pitch y Roll con FSF (2)**

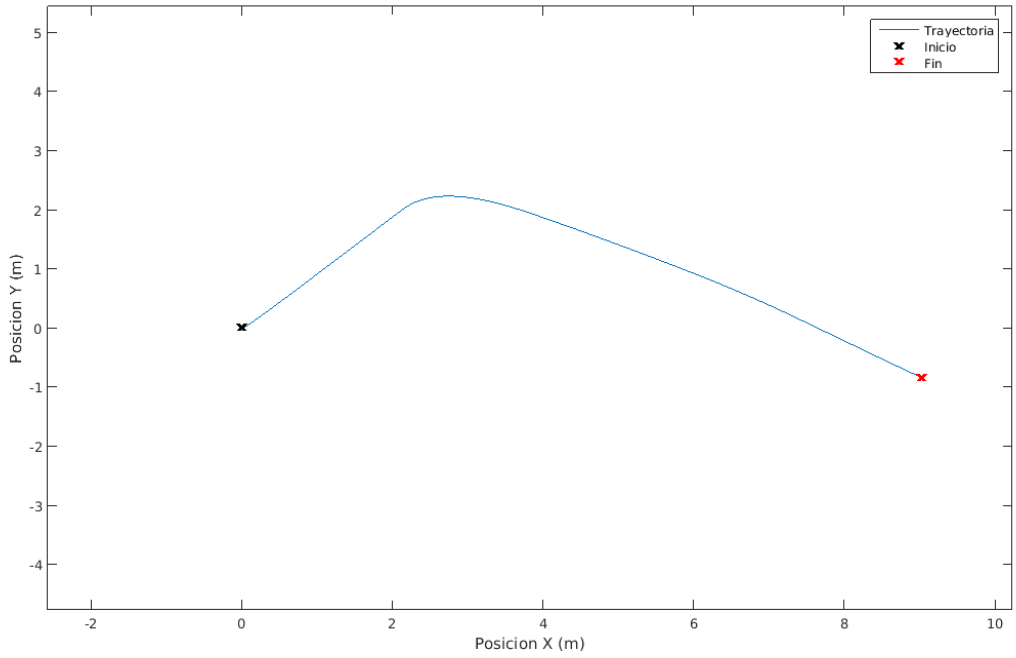



**Figura 4-39 Control de Yaw, Pitch y Roll con FSF , Y-X**

Siguiendo en la línea de los apartados anteriores, se realiza a continuación este mismo experimento en un vuelo real del drone (Figura 4-40).

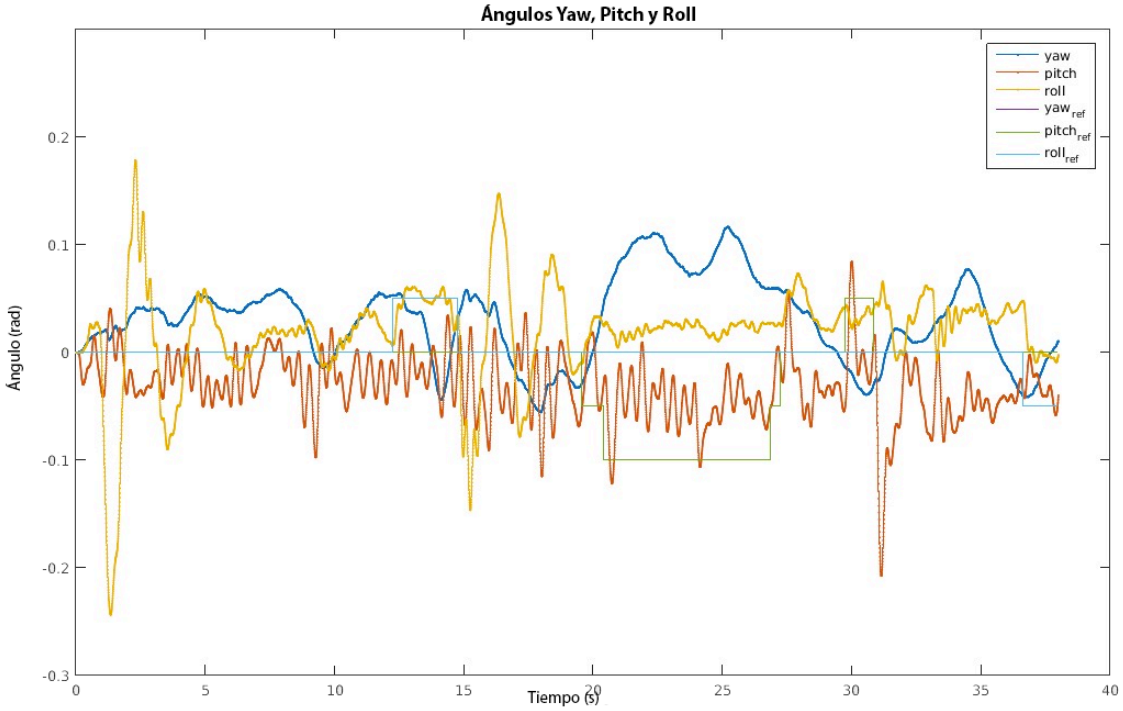



**Figura 4-40 Control de Yaw, Pitch y Roll en drone con FSF**

Con este controlador se tiene un despegue más brusco, ya que los polos que se utilizan para la posición en el eje Z le dan mucho peso para un control rápido, pero este cambio es corregido a los pocos segundos. Al igual que los otros controladores analizados, este tiene una serie de oscilaciones en todos los ángulos de Euler, pero es capaz de mantener mejor la posición indicada y seguir los cambios de referencia durante un mayor tiempo sin acabar en un choque. Comparando con los controladores LQR y PID se puede razonar que el controlador por asignación de polos tiene una respuesta mejor que el primero, más suave y con menos oscilaciones. En cuanto al segundo, el PID, en la gráfica se refleja un mejor funcionamiento pero si se ven los videos del repositorio de Github [2] se puede ver como el controlador por asignación de polos sigue de manera más fiel la referencia indicada.

Este experimento se corresponde con el video “Control FSF” en la carpeta “Videos” del repositorio [2].

### 4.3.3 Seguimiento de trayectoria marcada

Con este apartado se finaliza con los experimentos realizados para estos tres controladores escogidos, seguidamente se decidirá cual es sistema de control más óptimo para utilizar en el punto 5, donde se desarrollaran algunas aplicaciones interesantes para llevar más allá lo que el drone es capaz de hacer, dando pie también a futuros trabajos sobre este mismo vehículo.

En la Figura 4-41 se muestran los resultados para este último experimento.

A diferencia del resto de controladores, con este experimento se obtiene un resultado sin ninguna similitud al esperado. Tras varias modificaciones de los polos no se ha conseguido mejorar el funcionamiento del controlador en esta prueba.

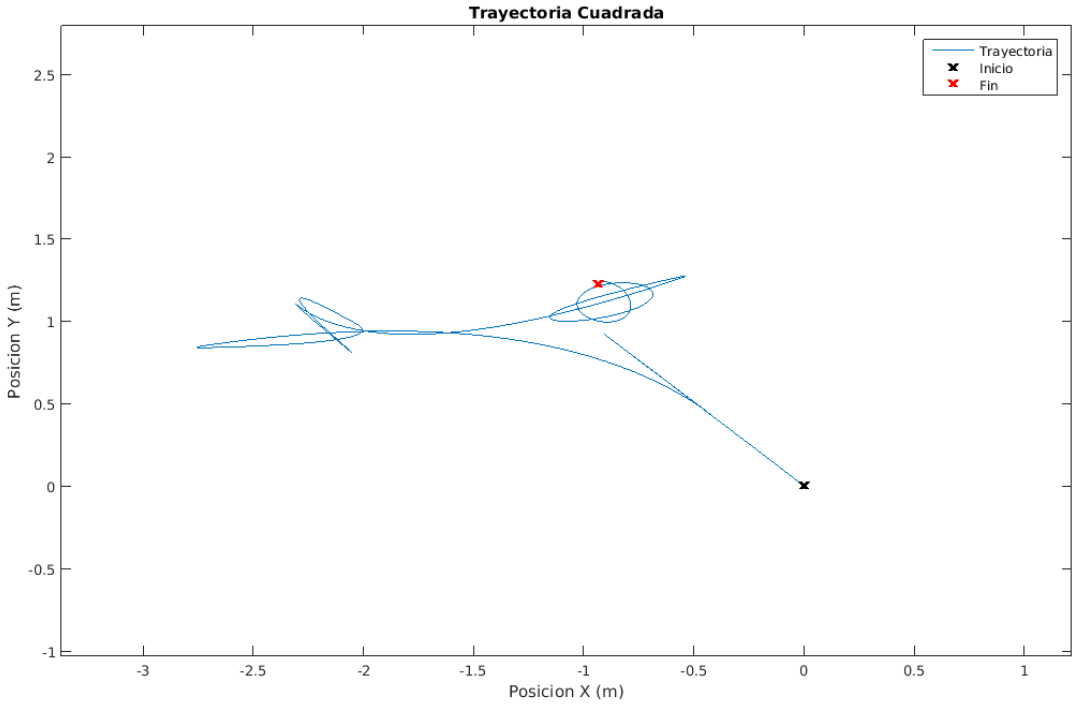



**Figura 4-41 Seguimiento de trayectoria FSF (1)**

Sin embargo, observando la Figura 4-42, donde se realiza la siguiente prueba, al igual que se hizo en los otros controladores, se ve una inmensa mejoría respecto al ensayo anterior. Con una distancia en módulo de 0.16 metros, es el controlador que ha conseguido dejar más cerca el punto de inicio y fin, del mismo modo, se ha reducido algo la distancia recorrida en el eje X.

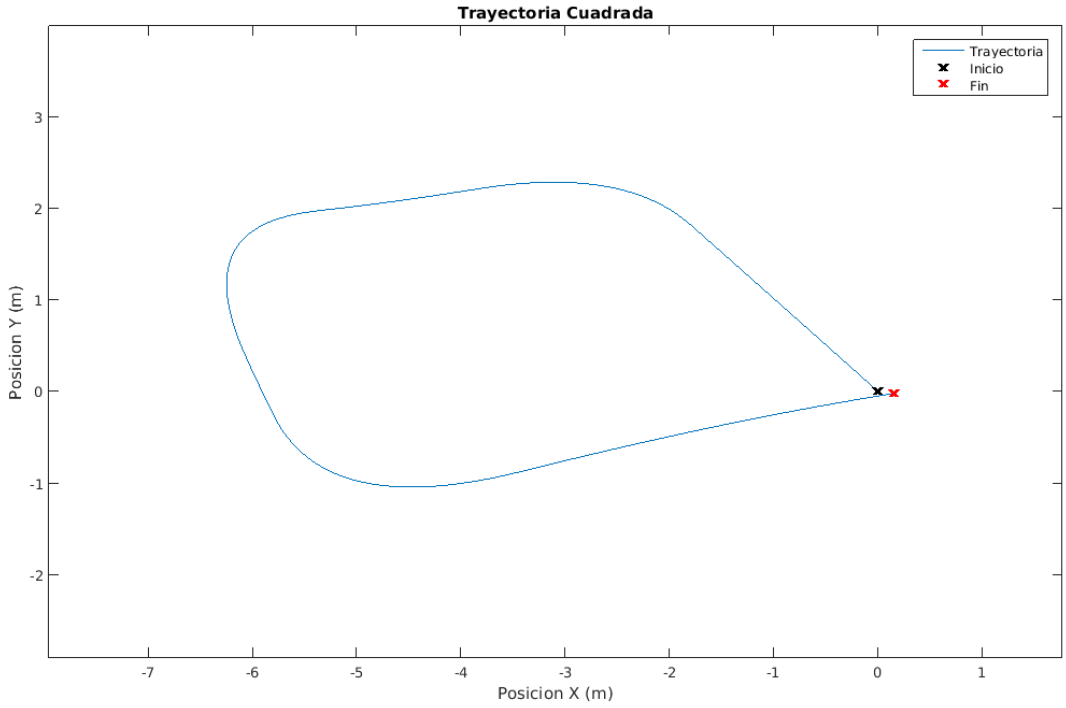



**Figura 4-42 Seguimiento de trayectoria FSF (2)**

## 4.4 Conclusión sobre controladores

Desués de este extenso desarrollo de los controladores elegidos y viendo los resultados obtenidos en los tres experimentos realizado con cada sitema de control, es necesario tomar el mejor de ellos para programar algunas aplicaciones que abran las fronteras de lo que el drone es capaz de hacer.

Cabe decir que cada drone posee sus ventajas e inconvenientes controlando un sistema tan complejo con es el estudiado. No hay ningún controlador que haya conseguido excelentes resultados en cada prueba, analizando en conjunto los resultados obtenidos:

- Control de altura → El controlador por asignación de polos tiene la mejor respuesta y más rápida.
- Control de yaw, pitch y roll → Tanto para pitch como para roll se obtienen resultados similares en los tres, pero el controlador PID tiene una mejor respuesta para el yaw que el resto.
- Seguimiento de trayectoria → Si se observa la primera prueba dentro de este experimento se tiene que el controlador LQR es el que funciona mejor, con una mejor figura y una mayor cercanía del punto final, sin embargo, observando la segunda prueba el controlador por asignación de polos trabaja mejor.

Por último, observando las representaciones de los resultados obtenidos de los vuelos reales del drone, no hay duda que el controlador por asignación de polos tiene un funcionamiento mejor que el resto de los estudiados, sin llegar a ser completamente perfecto proporciona un control más fiable que los otros dos controladores estudiados.

# 5 APLICACIONES

Ya se han analizado tres estructuras de control y se han hecho múltiples experimentos con ellas, se han podido observar las ventajas e inconvenientes de cada una. Llegados a este punto, se ve como una propuesta interesante el desarrollar algunas aplicaciones sobre el controlador que ha dado mejor resultado, el controlador por asignación de polos. Aunque en algunos de los experimentos se puede ver una mejor respuesta en la estructura PID o LQR, el controlador elegido les super con creces a la hora de rendimiento de vuelo real. Viendo únicamente las gráficas de vuelo real del apartado anterior no hay duda que este debe ser el controlador sobre el que desarrollar aplicaciones.

Si bien los sistemas de control son la base para cualquier vehículo aéreo, estos deben de comportarse de forma predecible y siguiendo las referencias ordenadas, lo interesante y la razón de la elección de este proyecto es el avance en nuevas habilidades del drone. Controlar el drone desde un terminal es cosa del pasado, ahora se quiere que el vehículo realice acciones por si mismo, que sea autónomo, que ayude en alguna acción al usuario que lo compro, por ejemplo, el caso de un drone capaz de fijar un objetivo en movimiento para seguirlo de forma autónoma para grabar algún deporte de forma sencilla [28].

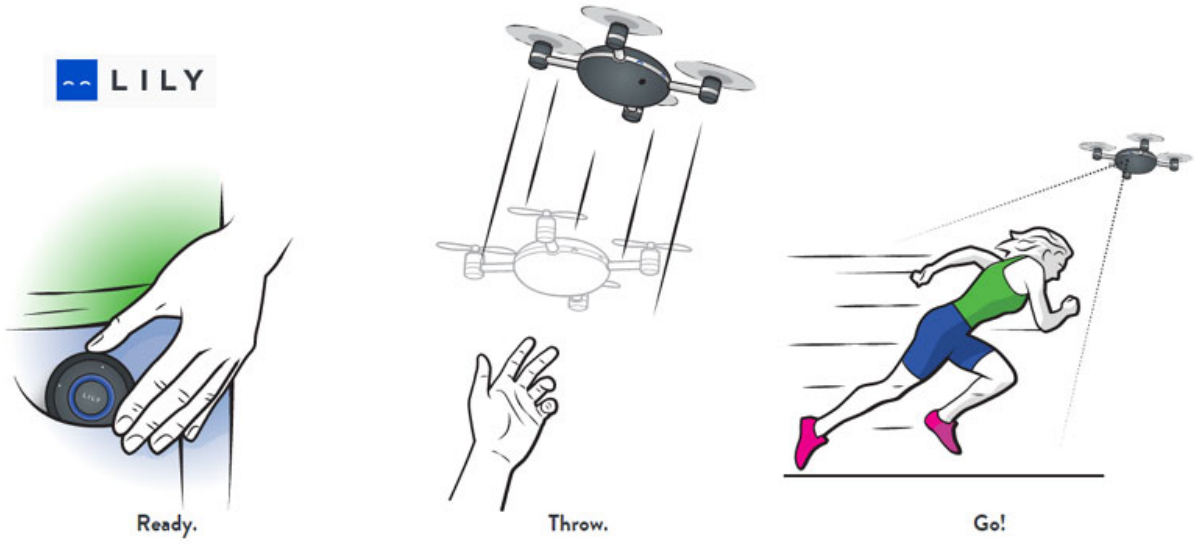



**Figura 5-1 Lily drone, capaz de seguir un objetivo en movimiento** [28]

Existe un gran abanico de posibles aplicaciones a desarrollar sobre este drone, teniendo en cuenta la precisión de la que se cuenta con el controlador utilizado y sin querer llegar a complicar demasiado este proyecto, se han escogido dos que ayudan a conocer más habilidades que este drone puede aportar y que dan pie a futuro trabajo sobre el vehículo. En concreto se han elegido: control por voz y localización de zona de aterrizaje con descenso progresivo.

## 5.1 Control por voz

Como primera aplicación se ha escogido el reconocimiento de comandos a través del habla, se puede coincidir en que existe un claro auge en las investigaciones acerca de esta temática, incluso las empresas más grandes

están apostando por esta tecnología para acercarla cada vez más a la vida cotidiana. Desde asistentes personales que leen los próximos eventos en el calendario y dan los datos sobre el último partido de fútbol hasta juguetes que se comunican con los niños para proporcionar un aprendizaje interactivo, no cabe duda de que el futuro rotará en torno a esta tecnología.

El objetivo que se pretende alcanzar es la reconocimiento de ordenes de movimiento específicas, es decir, las mismas funcionalidades que si se controlara el drone desde el teclado del ordenador directamente. Cabe decir que existen dos enfoques que se pueden utilizar para esta aplicación. El primero consiste en la adquisición de archivos de voz con *Matlab*, su propio análisis a partir de múltiples algoritmos y la comparación de la onda obtenida con una librería de archivos con las palabras más probables a ser dichas. Como se puede entender, si se tomara este enfoque se podría realizar un documento completo acerca del funcionamiento de este: rendimientos obtenidos, algoritmos usados y como mejorarlo. La otra opción, la elegida para este proyecto, es utilizar una librería externa que se encargue de todo ese proceso mencionado, la librería escogida pertenece a una gran compañía con fama en este sector, *Google.*

*Google* pone a disposición de los desarrolladores un gran número de herramientas, *APIs*[2]*,* para ser usadas de a partir de un coste que depende del uso, librándose así de todo el trabajo que conllevaría conseguir tales herramientas con un rendimiento tan bueno como aseguran. En concreto, para este proyecto se hace uso de la que se encarga del reconocimiento de voz, *Google Speech Cloud* [29]. Esta librería permite la conversión de audio en texto a partir de potentes redes neuronales, reconociendo más de 80 idiomas. El precio de uso es:

- Hasta 60 minutos, gratuito.
- A partir del minuto 61 tendría un precio de 0.006$ por cada 15 segundos de conversión.

Para el uso de esta herramienta se permite el uso de varios lenguajes de programación, para este proyecto se va a utilizar Python. Como lo que se quiere conseguir es que se puedan realizar las mismas acciones por voz que desde el teclado se relaciona el archivo en python con el código del control manual, *ReferenceValueServer.c*, que lee las teclas presionadas y envía las referencias al archivo de control, *rsedu_control.c*. Para ello una vez el drone comienza a volar se inica un hilo dedicado a recibir el comando reconocido, en este hilo nuevo se crea un socket TCP/IP en el puerto 55555 y permanece en escucha hasta que otro programa pida acceso, ese programa es ***reconocimientoVoz.py***, una vez se conecte se mantiene en espera hasta que reciba el comando reconocido, modificando así la referencia deseada y enviándola a través del socket inicial con *rsedu_control.c* para que lo aplique en el controlador.

Entrando en detalle del funcionamiento del programa Python, en primer lugar, se realiza la conexión con el puerto mencionado, seguidamente comienza el bucle principal. Se espera a que el usuario presione una tecla, indicando que desea hablar para reconocer el comando, se habla y el programa te devuelve la palabra reconocida, en caso de ser detectada, y se obtiene comando asociado a ella, siendo este enviado por el socket.

La función principal de reconocimiento, de la librería utilizada, recibe cuatro parámetros de entrada:

- Audio: archivo de audio grabado.
- Credenciales: archivo en formato json con los datos de la cuenta de *Google* utilizada, se obtiene desde la página web del servicio.
- Idioma: lenguaje que se quiere reconocer.
- Palabras: conjunto de palabras más probables de aparecer en el audio enviado.

[2] Interfaz de programación de aplicaciones (*Application programming interface*)

El archivo *reconocimientoVoz.py* se puede encontrar en la carpeta "Archivos Importantes" del repositorio de Github [2].

## 5.2 Localización de plataforma de aterrizaje

La elección de esta aplicación es un paso más hacía un vehículo autónomo, el objetico es detectar una señal en forma de X de color negro que marca la zona de aterrizaje del vehículo, una vez encontrada el vehículo deberá estabilizarse encima de la señal y proceder a descender de forma progresiva, para evitar oscilaciones en los controladores, todo ello de forma autónoma. Por tanto, el análisis de imagen esta enfocado a colores en lugar de a figuras, por simplificar algo el proceso. Se supone que la señal se encuentra pegada en un suelo de color claro de forma que no tiene ningún objeto oscuro alrededor.

Como ya se ha mencionado en el apartado 3.3, el análisis de imágenes se lleva a cabo dentro del archivo *rsedu_vis.c.* Teniendo el parámetro adecuados activado, *FEAT_POSVIS_RUN,* se activa el estudio de la imagen, aunque el drone hace 60 imágenes por segundo, únicamente se analizan 4 por segundo. Cada fotografía tiene un tamaño de 80x120 en términos de dos píxeles consecutivos, por lo que las matrices de la imagen tienen 120 filas y 80 columnas.

Cada imagen esta formada por tres matrices que constituyen las tres capas del formato YUV, aún así estas matrices pueden ser convertidas a formato RGB a partir de las siguientes ecuaciones [30]:

$$R = 1.164(Y - 16) + 1.596(V - 128)$$

$$G = 1.164(Y - 16) - 0.391(U - 128) + 0.813(V - 128)$$

$$B = 1.164(Y - 16) + 2.018(U - 128)$$

Con estas fórmulas se puede ir obteniendo el valor de cada píxel YUV en formato RGB a medida que se va recorriendo la imagen.

A continuación, se hace una media entre los tres valores RGB, obteniendo un valor en escala de grises, siendo el negro el valor 0 y el blanco el 255. Este dato se compara con un umbral predefinido, en este caso es 40, para decidir si se encuentra ante un píxel oscuro o no. En el caso de que está decisión sea positiva se incrementa una variable asociada con el área de píxeles oscuros en la imagen.

A parte de calcular el área, es buena idea detectar más características de la imagen que se esta analizando, por ejemplo el centro de gravedad. Para ello, se utilizan los momentos [31], estos son utilizados para la extracción de características relacionadas con el tamaño y forma de una imagen.

$$m_{r,s} = \sum_{i=1}^{M} \sum_{j=1}^{N} i^r j^s p_{i,j}$$

Con esta ecuación se calculan los momentos deseados, según r y s, se toma una imagen MxN, i hace referencia a las filas, j a las columnas y p es una plantilla del objeto. Esta plantilla es una matriz binaria de las mismas componentes que la imagen que tiene un 1 en los puntos que forman parte de la figura que se está intentando detectar y un 0 en lo demás. En este caso, se tendrá que la componente p(i,j) será 1 cuando el nivel de gris del píxel analizado sea inferior al umbral.

El momento $m_{00}$ hace referencia al área de la figura, $m_{10}$ y $m_{01}$ equivale a la distribución del área según filas y columnas. Por lo que durante el análisis de la imagen se calculan estos tres parámetros. Combinándolos se

puede obtener el centro de gravedad de la figura.

$$i_g = \frac{m_{10}}{m_{00}}$$

$$j_g = \frac{m_{01}}{m_{00}}$$

Llegados a este punto se tiene el área y el centro de gravedad de objeto a detectar, en el caso de que lo hubiera. Queda decidir si lo detectado se corresponde con lo que se busca, para ello se pregunta si el área detectada es mayor a otro umbral, 120. Como se supone que no hay objetos oscuros alrededor de la zona de aterrizaje el área calculada debería ser mínima hasta que la encuentre, por lo que no se activaría el descenso del drone.

Una vez el drone se encuentra justo encima de la plataforma se restablecen los ángulos de orientación de forma que el drone deja de avanzar y comienza a estabilizarse. Una idea que puede parecer curiosa es la de hacer pequeños movimientos con el vehículo hasta conseguir que el centro de gravedad de la figura se acerque al centro de la imagen, para lograr así que el drone esté justo encima de la señal, lo que se plantea es lo siguiente

- Calcular la distancia del centro de gravedad al centro de la imagen para los ejes X e Y, es decir, el error cometido.
- Aplicar un controlador PI para el roll y otro para el pitch, de forma que este error cometido va reduciéndose hasta llegar cerca del valor deseado.
- Comenzar descenso.

El problema con esta idea es que los controladores utilizados no llegan a tener la precisión de centímetros que haría falta para una tarea así, por lo que en lugar de centrar el drone con la plataforma se conseguirían oscilaciones que podrían hacer que el vehículo se alejara de la figura.

Siguiendo con el funcionamiento logrado, una vez detectada la plataforma se activaría una variable que indica el deseo de iniciar el descenso. Esto es recibido por el programa de control *rsedu_control.c* que espera unos segundos para que el drone se estabilice y comienza el descenso progresivo. Este aterrizaje consiste en ir reduciendo la altura en intervalos de 15 centímetros, evitando así que el drone pierda la estabilidad.

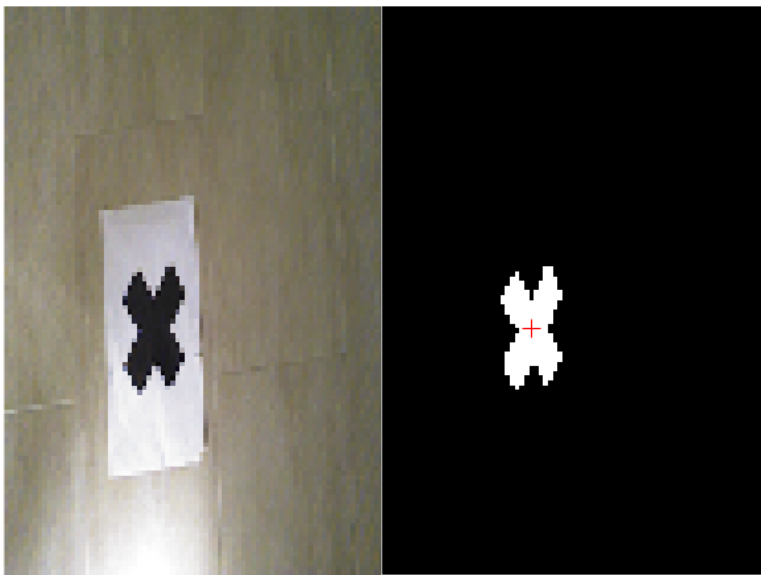


**Figura 5-2 Proceso de detección de plataforma.**

Por último, en la Figura 5-2 se puede ver un ejemplo práctico de cómo vería el drone la zona de aterrizaje y cómo la procesaría para decidir si se corresponde con lo que se busca o no. La cruz roja de la imagen de la derecha se corresponde con el centro de gravedad de la señal.

# 6 CONCLUSIONES Y LÍNEAS FUTURAS

## 6.1 Conclusiones

Los objetivos de este proyecto ha sido formar las bases de funcionamiento de la librería diseñada por el *MIT* para el minidrone utilizado de forma que pueda ser utiizada en futuros trabajos, diseñar controladores para sistemas complejos y desarrollar aplicaciones sobre estos sistemas. Por lo que se puede acordar que se ha conseguido satisfactoriamente.

Se ha logrado reunir toda la información acerca del Toolbox mencionado en un único documento de forma detallada, los archivos de código más importantes y las partes de estos más esenciales para modificar. Se ha podido ver las características del quadrotor y las ventajas e inconvenientes que posee.

Se han diseñado tres sistemas de control: PID, LQR y por asignación de polos. Se han podido ver los resultados de estos controladores ante diferentes situaciones, observando cual tenía mejor respuesta. Se llegó a la conclusión de que el controlador por asignación de polos era el que mejor funcionaba en general, sobre todo teniendo en cuenta los resultados de vuelo real.

Por último, se han desarrollado dos aplicaciones interesantes sobre el controlador elegido. Por un lado, reconocimiento de comandos de voz con la librería de *Google*, logrando que el drone realizara las mismas acciones que si fuera controlado de forma manual desde el teclado. Por otro lado, se ha utilizado la cámara que incorpora el vehículo para analizar las imágenes que toma en tiempo real y reconocer una plataforma de aterrizaje.

A pesar de los numerosos problemas que se han tenido durante la realización del proyecto: problemas de conexión con el drone, controladores no reaccionando de la forma esperada, daños en las hélices que modificaban la dinámica del drone, etc. Se ha podido superar y seguir avanzando hasta conseguir el resultado aquí presente.

## 6.2 Líneas futuras

Para concluir este documento sería interesante mencionar algunas mejoras que se podrían realizar para conseguir un funcionamiento más adecuado e ideas de algunas aplicaciones más complejas:

- En cuanto a los controladores, sería buena idea continuar la investigación y probar algunas estructuras de control adicionales que se han aprendido en la universidad, como por ejemplo el predictor de Smith o un control adaptativo.
- Control del drone mientras realiza alguna acrobacia, 360º sobre si mismo en pitch o roll.
- Aterrizaje progresivo en plataforma en movimiento, al ir modificando su posición el drone deberá intentar mantener el centro de gravedad en el centro de la imagen, siguiendo así el movimiento que esté realizando la plataforma.
- Continuar con la investigación del análisis de imágenes, detectando figuras en lugar de colores o códigos QR que contengan la acción a realizar en su interior.

# 7 BIBLIOGRAFÍA

[1] The MathWorks, Inc, «Mathworks,» [En línea]. Available: https://es.mathworks.com/help/simulink/. [Último acceso: 22 Febrero 2017].

[2] E. Marín, «Repositorio del proyecto,» [En línea]. Available: https://github.com/emiliomarin/TFG-ParrotRollingSpider.

[3] A. Barrientos, J. del Cerro, P. Gutiérrez, R. San Martín, A. Martínez, C. Rossi, Vehículos aéreos no tripulados para uso civil. Tecnología y aplicaciones, Madrid, 2007.

[4] E. Darack, «Air & Space Magazine,» [En línea]. Available: http://www.airspacemag.com/photos/a-brief-history-of-unmanned-aircraft-174072843/. [Último acceso: 26 Febrero 2017].

[5] Timothy H. Cox, Christopher J. Nagy, Mark A. Skoog, and Ivan A. Somers, «Civil UAV Capability Assessement,» [En línea]. Available: https://www.nasa.gov/centers/dryden/pdf/111761main_UAV_Capabilities_Assessment.pdf. [Último acceso: 26 Febrero 2017].

[6] N. Civil UAV Assessment Team, «Earth Observations and the Role of UAVs,» [En línea]. Available: https://www.nasa.gov/centers/dryden/pdf/175939main_Earth_Obs_UAV_Vol_1_v1.1_Final.pdf. [Último acceso: 26 Febrero 2017].

[7] C. Perley, «Improvement in discharging explosive shells from balloons». EEUU Patente US37771 A, 24 Febrero 1863.

[8] Parrot, «Rolling Spider,» [En línea]. Available: http://global.parrot.com/au/products/rolling-spider/. [Último acceso: 02 03 2017].

[9] Parrot, MiniDrones Rolling Spider User Guide.

[10] Parrot, «FreeFlight Mini para Android,» [En línea]. Available: https://play.google.com/store/apps/details?id=com.parrot.freeflight4mini&hl=en. [Último acceso: 04 03 2017].

[11] Parrot, «FreeFlight Mini iPhone,» [En línea]. Available: https://itunes.apple.com/us/app/freeflight-mini/id1137022728?mt=8. [Último acceso: 04 03 2017].

[12] Ubuntu, «Ubuntu,» [En línea]. Available: https://www.ubuntu.com/download/alternative-downloads. [Último acceso: 14 Marzo 2017].

[13] VMWare, Inc, «VMWare,» [En línea]. Available: http://www.vmware.com. [Último acceso: 14 Marzo 2017].

[14] R. S. G. B. Yogianandh Naidoo, «Quad-Rotor Unmanned Aerial Vehicle Helicopter Modelling & Control,» *International Journal of Advanced Robotic Systems,* vol. 8, nº 4, 2011.

[15] 16.30/31 Feedback Control Systems MIT Course, «LQR Control - Drone Dynamics,» [En línea].

Available: http://peris.mit.edu/1630/labs/lab_03%20-%20LQR/1_pset_03/pset_03.pdf.

[16] Parrot-Developers. [En línea]. Available: https://github.com/Parrot-Developers/RollingSpiderEdu. [Último acceso: 20 Marzo 2017].

[17] «16.30 Feedback Control Systems,» [En línea]. Available: http://fast.scripts.mit.edu/dronecontrol/. [Último acceso: 20 Marzo 2017].

[18] Parrot, «Firmware comercial 1.99.2,» [En línea]. Available: http://drivers.softpedia.com/get/drone-helicopter-multicopter-controller/Parrot/Parrot-Rolling-Spider-Drone-Firmware-1992.shtml. [Último acceso: 25 Marzo 2017].

[19] G. W. y. G. Bishop, «An Introduction to the Kalman Filter,» 2001. [En línea]. Available: https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf.

[20] P. Corke, Robotics, Vision & Control, Springer, 2017.

[21] J. F. Melero, «Sincronización y comunicación por paso de mensajes,» de *Apuntes de la asignatura "Informática Industrial"*.

[22] M. R. Arahal, «Representación del color,» de *Apuntes de la asignatura "Sistemas de Percepción"*, 2017.

[23] K. J. A. y. T. Hägglund, Control PID avanzado, Pearson - Prentice Hall, 2009.

[24] K. Ogata, Ingeniería de control moderna 5ª Edición, Pearson, 2010.

[25] T. Á. Cantero, «Diseño de controladores discretos,» de *Apunter de la asignatura "Ingeniería de Control"*.

[26] J. P. H. y. E. Frazzoli, «Deterministic LQR,» de *16.30/31 Feedback Control Systems MIT Course*.

[27] J. P. H. y. E. Frazzoli, «Full-State Feedback Control,» de *16.30/31 Feedback Control Systems MIT Course*.

[28] «Lily Drone,» [En línea]. Available: https://www.lily.camera.

[29] Google, «Google Speech Cloud,» 25 Agosto 2017. [En línea]. Available: https://cloud.google.com/speech/?hl=es.

[30] FOURCC, «FOURCC Codecs,» [En línea]. Available: https://www.fourcc.org/fccyvrgb.php. [Último acceso: 10 Agosto 2017].

[31] M. R. Arahal, «Separación de regiones,» de *Apuntes de la asignatura "Sistemas de Percepción"*.

[32] D. R. R. y. T. Á. Cantanero, «Diseño de controladores discretos,» de *Apuntes de la asignatura "Ingeniería de control"*.